DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943-5101

REPORT DOCUMENTATION PAGE

EPORT SECURITY CLASSIFICATION JNCLASSIFIED	1b. RESTRICTIVE MARKINGS
ECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT
ECLASSIFICATION/DOWNGRADING SCHEDULE	Approved for public release; distribution is unlimited
RFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
AME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b. OFFICE SYMBOL (if applicable) CS
	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
DDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
AME OF FUNDING/SPONSORING RGANIZATION	8b. OFFICE SYMBOL (if applicable)
	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
DDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION N

ITLE (Include Security Classification)

A Model and Algorithms for a Software Evolution Control System

ERSONAL AUTHOR(S)

LTC Salah El-Din Mohammed Badr

TYPE OF REPORT	13b. TIME COVERED		14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Ph.D. Dissertation	From	To	December 1993	464

UPPLEMENTARY NOTATION

The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
ELD	GROUP	SUB-GROUP	software evolution, software evolution steps, configuration graph, version control configuration management,

BSTRACT (Continue on reverse if necessary and identify by block number)

This dissertation introduces an Evolution Control System (ECS) for the Computer Aided Prototyping Syste
'S. The purpose of the ECS is to automate the scheduling and the assignment of tasks to the software designe
d on management policies and the dependencies in a model of the software configuration. The ECS controls t
vare evolution process in an incrementally evolving software system where the steps to be scheduled are on
ally known. Time required, the set of sub-tasks for each step, and the input/output constraints between steps a
ncertain, and are all subject to change as evolution steps are carried out. The ECS provides computer assistan
nanaging such changes and partially automates the control of the design team and the project data.

The ECS manages both the development/prototyping data and the design team through scheduling the softwar

ISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
NAME OF RESPONSIBLE INDIVIDUAL Valdis Berzins	22b. TELEPHONE (Include Area Code) (408) 656-2461
	22c. OFFICE SYMBOL Code CS/Be.

ORM 1473, JUN 86 *Previous editions are obsolete*

tasks and assigning them to members of the design team. The main goals of this system are: 1. Managing the evol steps from the moment they are proposed until their completion. 2. Reaching a feasible schedule that meets the line requirements or minimizes the largest amount that a deadline is missed if all deadlines cannot be met and pro for the earliest possible completion for those steps that either do not have deadlines or have under-estimated deadl 3. Maximizing the efforts of software designers by maximizing concurrent assignments. 4. Supporting incremen planning as additional information becomes available. 5. Minimizing wasted design effort due to schedule reorga tion as well as workers forced to wait for completion of sub-tasks. 6. Insuring system integrity via propagation change consequences (induced steps) to maintain the global consistency of the database and providing serializal of updates. 7. Efficient use of space and time for the design database and scheduling algorithm. 8. Automating th cess of determining which versions of the subcomponents belong to each version of the entire system. 9. Provid computer assistance for task tasks and assigning them to members of the design team. The main goals of this sy are: 1. Managing the evolution steps from the moment they are proposed until their completion. 2. Reaching a fea schedule that meets the deadline requirements or minimizes the largest amount that a deadline is missed if all dead cannot be met and provides for the earliest possible completion for those steps that either do not have deadlines or under-estimated deadlines. 3. Maximizing the efforts of software designers by maximizing concurrent assignmer Supporting incremental replanning as additional information becomes available. 5. Minimizing wasted design e due to schedule reorganization as well as workers forced to wait for completion of sub-tasks. 6. Insuring system rity via propagation of change consequences (induced steps) to maintain the global consistency of the database providing serializability of updates. 7. Efficient use of space and time for the design database and scheduling al rithm. 8. Automating the process of determining which versions of the subcomponents belong to each version o entire system. 9. Providing computer assistance for task

The proposed ECS system represents a management layer between the user interface (supporting two user cl managers and designers) and the design database which contains a record of the versions of all software objec planned, active and completed evolution steps.

Approved for public release; distribution is unlimited

A Model and Algorithms for a Software Evolution Control System

by

Salah El-Din Mohammed Badr
LTC, Egyptian Army
B.S., Military Technical College, Cairo, Egypt 1976
M.S.C.E., Faculty of Engineering, Cairo University, 1988

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1993

C.1

ABSTRACT

This dissertation introduces an Evolution Control System (ECS) for the Computer Aided Prototyping System CAPS. The purpose of the ECS is to automate the scheduling and the assignment of tasks to the software designers based on management policies and the dependencies in a model of the software configuration. The ECS controls the software evolution process in an incrementally evolving software system where the steps to be scheduled are only partially known. Time required, the set of sub-tasks for each step, and the input/output constraints between steps are all uncertain, and are all subject to change as evolution steps are carried out. The ECS provides computer assistance for managing such changes and partially automates the control of the design team and the project data.

The ECS manages both the development/prototyping data and the design team through scheduling the software tasks and assigning them to members of the design team. The main goals of this system are: 1. Managing the evolution steps from the moment they are proposed until their completion. 2. Reaching a feasible schedule that meets the deadline requirements or minimizes the largest amount that a deadline is missed if all deadlines cannot be met and provides for the earliest possible completion for those steps that either do not have deadlines or have under-estimated deadlines. 3. Maximizing the efforts of software designers by maximizing concurrent assignments. 4. Supporting incremental replanning as additional information becomes available. 5. Minimizing wasted design effort due to schedule reorganization as well as workers forced to wait for completion of sub-tasks. 6. Insuring system integrity via propagation of change consequences (induced steps) to maintain the global consistency of the database and providing serializability of updates. 7. Efficient use of space and time for the design database and scheduling algorithm. 8. Automating the process of determining which versions of the subcomponents belong to each version of the entire system. 9. Providing computer assistance for task

NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

decomposition during planning using decomposition and dependency information of the previous version of the software system.

The proposed ECS system represents a management layer between the user interface (supporting two user classes, managers and designers) and the design database which contains a record of the versions of all software objects and planned, active and completed evolution steps.

TABLE OF CONTENTS

I. INTRODUCTION 1

A. PURPOSE 1

B. SOFTWARE EVOLUTION 2

C. SOFTWARE EVOLUTION MODELS 2

D. PROTOTYPING PROCESS 3

E. PROBLEM DEFINITION 3

F. CONTRIBUTION OF THIS RESEARCH 4

G. ORGANIZATION OF CHAPTERS 5

II. SURVEY OF RELEVANT WORK 6

A. OVERVIEW 6

B. REVIEW OF FORMAL EVOLUTION MODELS 6

C. VERSION CONTROL AND CONFIGURATION MANAGEMENT 8

D. TOOL SUPPORT FOR CONFIGURATION MANAGEMENT AND COOPERATIVE DESIGN 11

E. APPROACHES TO SCHEDULING EVOLUTION STEPS 13

1. Scheduling Tasks with Precedence Constraints 15

III. REQUIREMENTS ANALYSIS 18

A. GOALS AND JUSTIFICATIONS 18

B. GRAPH MODEL OF SOFTWARE EVOLUTION 19

1. Summary of Modifications to the Original Graph Model 23

a. Version and Variation Numbering 24
b. Configuration Management 26
2. States of Evolution Steps 29
a. Proposed State 29
b. Approved State 31
c. Scheduled State 31
d. Assigned State 32
e. Completed State 32
f. Abandoned State 32
g. Relation to the Original Graph Model 33
3. Constraints on State Transitions 33
4. Specifying Inputs to the Evolution Steps 34
5. Induced Evolution Steps 34
6. Induced State Transitions 36
7. Applying The General Graph Model to PSDL 36
C. SCHEDULING MODEL 39
1. Scheduling Constraints 39
2. Dynamics - What Can Change 41
a. Primary Input Changes 42
b. Secondary Input Changes 42
c. Affected Modules Changes 43

d. Precedence, Priority, and Deadline Changes 43
e. Step Decomposition 44
f. Time Estimate Changes....46
g. Designer-Pool Changes....47
h. Impact of Changes....47
3. Computational Feasibility 47
D. RESOURCE MODEL 48
E. ECS MODEL....48
1. Context Model....48
2. Event List 49
3. State Model And Related Concepts 50
a. Configuration graph 50
b. Designer_Pool....51
c. Schedule 51
4. Manager Interface 51
5. Designer Interface 52
F. VALIDATION OF ECS SPECIFICATION....56
1. A Typical Scenario....56
2. Scenario for a Missed Estimated Finish Time 66
3. Scenario for an Early Commit....66
4. Scenario for Changing the Precedence of a Step 67

a. Changing Precedence Leading to an Infeasible Schedule....68

b. Changing Precedence Leading to Infeasible Schedule and Step Suspension....68

5. Scenario for Changing the Decomposition of an Assigned Task....69

6. Assessment of the Adequacy of the Proposed Command Set....71

7. Questions and Design Decisions....72

IV. DESIGN DEVELOPMENT OF ECS....74

A. MODELING THE DESIGN DATABASE....74

1. Design Database Schema....74

a. Type Object....75

b. Type Version....76

c. Type Component....77

d. Type Top_component....77

e. Type Step....78

f. Type Top_step....80

g. Type Designer....81

h. Type Assignment....82

i. Type Schedule....82

j. Type Sequencer....83

k. Type Text_Object....83

2. Concurrency Control....84

B. IMPLEMENTATION CONSIDERATIONS....86

1. Implementing Shared Data for Multiple Users 86
a. Shared Data Space 86
b. Private Workspaces 87
2. Choice of Languages and Support Systems 88
3. Software Decomposition and Structure 88
C. THE SCHEDULING PROBLEM 88
1. The Scheduling Algorithm 89
a. System and Task Model 90
b. A Heuristic Search Scheduling Algorithm 91
c. Algorithm for Adjusting Deadlines 97
V. EVALUATION AND VALIDATION 104
A. COMPLEXITY ANALYSIS 104
B. Simulation Study 104
1. Simulation Method 105
2. Simulation Results 107
C. DEVELOPMENT OF TEST CASES 109
1. Determining Change Consequences 110
2. Enforcing Change Consequences for Global Consistency 113
3. Incremental Planning 115
4. Changes in the Plan 118
a. Early Commit of a Step 118

b. Increasing Estimated Duration of a Step 120
c. Suspending a Step 120
d. Committing a Step 122
e. Dropping a Designer 124
D. ANALYSIS OF RESULTS 126
VI. CONCLUSIONS 127
A. SUMMARY 127
B. IMPORTANCE OF RESEARCH RESULTS 128
C. PROPOSED EXTENSIONS 128
VII. APPENDICES 130
A. Formal Specifications 130
1. State Model and related concepts 130
a. Configuration graph 133
b. Designer_Pool 137
c. Schedule 137
d. Assignments 138
2. Behavior Model 139
3. Manager Interface 150
4. Designer Interface 157
5. Type Time 158
B. DESIGN DATABASE SCHEMA 160

1. Class Step 160
2. Class Component 226
3. Class Designer 265
4. Class Assignment 272
5. Class Time 284
C. PROGRAMS 293
1. The Ada Interface to DDB Package 293
2. The Scheduler Package 313
3. Main Programs 349
D. TEST DATA AND TEST RESULTS 415
VIII. LIST OF REFERENCES 441
INITIAL DISTRIBUTION LIST 448

ACKNOWLEDGMENTS

First and foremost, I must acknowledge the unfailing and unconditional support I have received through it all from my wife, Enas, and my daughters, Hala and Hoda, without which this work could never have been completed. Their positive attitude and understanding were remarkable as they were essential to my success.

I also wish to express my deepest gratitude to Professor Valdis Berzins whose support, guidance, knowledge and enthusiasm have been a constant inspiration to me. Special thanks and deep gratitude are also due to Professor Luqi and Professor Shing for their continuous support and guidance.

Finally, I wish to thank the other members of my committee for their support and special insights; the Computer Science Department staff for their outstanding assistance; and my fellow Ph.D. students who always provided encouragement and comic relief when times got hard.

I. INTRODUCTION

A. PURPOSE

The main objective of this dissertation is to design an evolution control system that can provide automated assistance for the software evolution process in an uncertain environment where designer tasks and their properties are always changing. The software evolution process involves: 1) The software users (customers) who initiate the change requests whether they are corrective, adaptive, or perfective changes [3], 2) The software manager or change control board who reviews these change requests and approves/rejects the changes for implementation, 3) The evolution team that has the task of implementing/ verifying these changes, 4) and finally the software system under evolution that must preserve its consistency and keep enough information about its evolution history.

We view an Evolution Control System (ECS) as the agent that keeps track of proposed, ongoing, and completed changes to a software system. It provides automated assistance to the software evolution manager to help him/her to make the right decisions. It automatically propagates change consequences by defining the set of possibly affected modules. It also coordinates and plans change implementation activities within the design team in a way that supports team work and guarantees system integrity, as well as adapting itself to the dynamic nature of the evolution process where new changes arrive randomly and current modifications are themselves subject to change as more information becomes available.

The above definition implies that an ECS has two main functions. The first is to control and manage evolving software system components (version control and configuration management, VCCM) and the second is to control and coordinate evolution team interactions (planning and scheduling software evolution tasks which we refer to as evolution steps).

The ECS system should manage both human resources and the design database and provide the help needed by the software manager as well as facilitating the designers' tasks. This system provides the required algorithms for coordinating and executing the activities mentioned above as well as the algorithms for reaching and maintaining a feasible schedule, if one exists, that meets the deadline requirements, reduces/avoids rollbacks, and

insures system integrity in an uncertain environment where the set of evolution steps and their properties are always changing.

B. SOFTWARE EVOLUTION

Software evolution is the process of extending or modifying the functionality of a software system [11]. Evolution activities may be triggered by changes in user requirements, planned phased development of a system, or by design changes to eliminate errors discovered after system delivery (repair). Evolution (maintenance) activities account for 65% to 75% of total cost of a software system [71]. Software evolution involves change requests, software systems, evolution steps, customers, managers, and software engineers. Customers provide change requests, and the corresponding changes are controlled by the managers of the software system. Approved changes trigger evolution steps that produce new versions of the software system. Evolution steps are scheduled based on management policies, and are executed by the software engineers [52].

C. SOFTWARE EVOLUTION MODELS

While detailed software process models are still a subject of research, there are some general models (paradigms) of software development that can be identified. Some of these software development models are[71]:

1. The waterfall model: This model views the software process as a cascade of a number of phases such as requirements, specification, design, implementation, testing and maintenance phase.
2. Exploratory programming: In this approach a working system is rapidly developed, then repeatedly modified until it reaches an adequate functionality. This model is used where detailed requirements cannot be specified and where adequacy rather than the correctness is the main goal of system designers.
3. Prototyping: This approach is similar to the exploratory programming, but the main goal is establishing the system requirements. This normally followed by an implementation of the requirements to obtain a production quality system.
4. Formal transformation: In this approach a formal specification of the system is developed then transformed to a program using correctness preserving transformations.
5. System assembly from reusable components: This approach uses the assumption that systems are mostly made up of already existing components. This means that the system development becomes an assembly rather than a creation process.

These models are normally called software life cycle models to cover the period from conception to retirement of a given software system. This means that the evolution activities can follow the same model followed in the development. In some of these models evolution starts after the release of the developed system, like the waterfall model in which the evolution activities go through the same cascaded phases of the model. In some other models like exploratory programming and evolutionary prototyping the current system can be viewed as a snapshot of an evolving system that evolves from an empty system through continuous iterations of evolution steps.

D. PROTOTYPING PROCESS

Prototyping is a technique to help establish and validate system requirements. Prototyping in the software process is practiced in two different forms; the first is "throw-away" prototyping where a prototype is developed with the objective of specifying system requirements. After the customer is satisfied with the requirements the prototype is discarded and the system is built from scratch.

The second form is evolutionary prototyping where a prototype evolves via a number of versions to the final system. Evolutionary prototyping lends itself as an evolution model where the system is started from its fundamental concepts and is then iteratively modified in an interactive way with the customer until the system reflects the customer's real needs [51].

Prototyping techniques include the use of executable specification languages and reusable software components for rapid prototype construction.

E. PROBLEM DEFINITION

With the complexity of software systems growing every day, more sophisticated development and maintenance environments are necessary to cope with the evolutionary nature of software systems. These systems experience iterative modifications through many versions to cope with the customer's changing and growing needs and the changing and growing software and hardware technology.

In the context of an evolving system, a software evolution step is used to represent the activities of analyzing and implementing one change request. These evolution steps are

typically only partially known. Time required, the set of sub-tasks for each step, and the input/output constraints among steps are all uncertain, and are all subject to change as evolution steps are carried out.

Scheduling these evolution steps without taking into account their special nature, as mentioned above, complicates the management task of achieving the best possible utilization of human and machine resources. This also may lead to software rollbacks which waste programming efforts, and affect software consistency due to the lost coordination between engineers working on different evolution steps that may be related to each other.

An evolution control system must account for all the interacting factors of the evolution process. These factors, as discussed above, include change requests that are provided by the customers and lead to the creation of corresponding evolution steps. These steps are controlled by the manager of the software system. Approved steps are scheduled for implementation by software designers. The completed steps produce new versions of the software system. Controlling an evolving system means coordinating these interactions in a way that preserves system integrity, supports team work via maximizing the number of concurrent assignments, avoids/reduces rollbacks, planning the required changes (steps) to meet the management constraints such as deadlines, precedences, and priorities, and maintaining a record of these change activities for history purposes. Such an evolution control system should be flexible enough to adapt its scheduling and planning function, in real-time, to the dynamic changes in current evolution steps as well as the random arrival of new steps.

F. CONTRIBUTION OF THIS RESEARCH

The main contributions of this dissertation are:

1. Automated support for changes in plan during the execution of the plan.
2. Automatic decision support for planning and team coordination based on design dependencies captured in the configuration model.
3. The enhancement and implementation of a configuration graph model presented in [52], which is used to keep the evolution history of software systems.

4. The development of the specification and implementation of the required mechanisms to manage the evolution steps from the moment a system is proposed until its completion.
5. The development of an automated version control and configuration management mechanism which is transparent to the users. This mechanism automatically determines the version and variation numbers of the software component versions and decides which component version belongs to which system configuration.
6. The development and implementation of a mechanism for detecting change consequences (determine the components affected by a change) to maintain the global consistency of the design database and provide serializability of updates for each variation.
7. The development and implementation of an on-line scheduling algorithm for finding a feasible schedule that: meets the deadlines and precedence constraints of all the active steps or suggests new deadlines for the lowest priority deadlines until a feasible schedule that meets the deadlines of the higher priority steps is reached. This algorithm also:
    a. Supports teamwork by concurrently assigning ready steps to available designers.
    b. Supports incremental replanning as additional information becomes available.
    c. Minimizes wasted design effort due to reorganization of the schedule as well as workers forced to wait for completion of sub-tasks via the immediate detection of new dependencies forced by this reorganization and the suspension and rescheduling of the affected assigned-steps.

G. ORGANIZATION OF CHAPTERS

The rest of this dissertation is organized as follows: Chapter II provides an overview of significant related research. Requirement analysis of the proposed system is given in Chapter III. Chapter IV discusses the design and development of the proposed evolution control system and our heuristic algorithm for scheduling the evolution steps as well as the algorithms for the rest of the system functions. The evaluation and validation of the proposed system is presented in Chapter V. Chapter VI includes the concluding remarks, evaluates the contribution of the dissertation, and provides directions for future work.

II. SURVEY OF RELEVANT WORK

A. OVERVIEW

The main areas in software engineering relevant to ECS are software development/ evolution, version control and configuration management, task planning and scheduling, and concurrency control. As defined in Chapter I, the ECS has two main functions. The first is to control and manage the evolving software system components which is directly related to the area of version control and configuration management, VCCM. The second is to control and coordinate the evolution team interactions that include coordinating their simultaneous access to the changing software components with the required concurrency control to guarantee system integrity, and coordinating and assigning their tasks in such a way that maximizes the concurrent assignment and meets management constraints such as deadlines and precedences.

B. REVIEW OF FORMAL EVOLUTION MODELS

In [52], Luqi presents a graph model for software evolution that introduced the notion of evolution step as the activities of initiation analysis and implementation of one request for change in the system under evolution. Luqi models the software system evolution history as an acyclic bipartite graph G = {C, S, I, O}. C nodes represent system components and S nodes represent evolution steps. The input edges I represent the relation between a step and the set of system components that have to be examined to produce output components which are consistent with the rest of the system. The output edges O represent the relation between an evolution step and the components it produces. The states of an evolution steps as well as the generation of substeps to propagate the change consequences are also defined. In this dissertation we extend this graph model to include other relations among system components ("part_of" and "used_by") and the "part_of" relationship between composite step and its substeps. Details of the graph model for software evolution and its extensions are presented in Chapter III.A as it is the basis for our system.

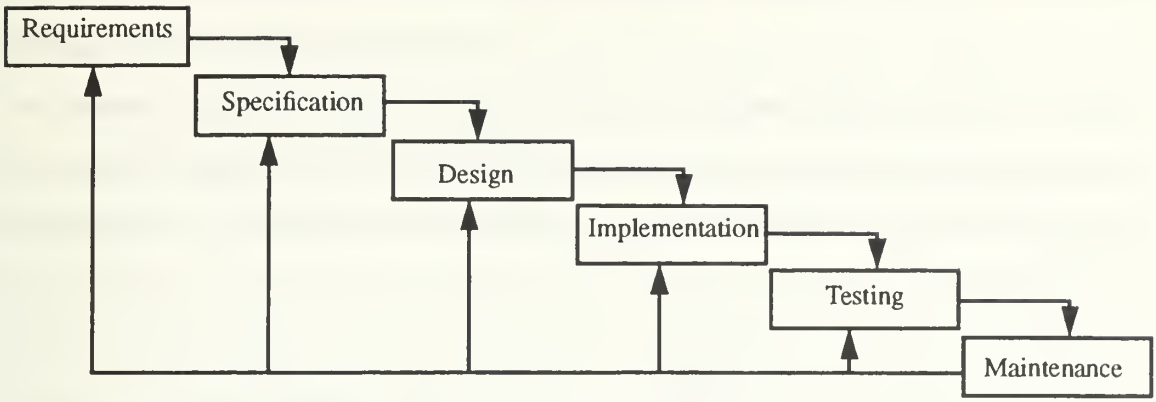


FIGURE 1. Waterfall model

The waterfall model as depicted in Figure 1 is a software life cycle model that covers the period from conception to retirement of a given software system. It is clear from Figure 1 that the evolution (maintenance) activities follow similar sequence of steps like those in the development. Unlike the original development cycle, the evolution activities (adaptive, corrective, and perfective maintenance) must take into consideration the existing system's requirements, decomposition, constraints, capabilities and performance. The effect of the changes must be propagated to preserve system consistency. In the mean time, concurrent changes must be coordinated to avoid rollbacks and wasting engineering effort. Evolution changes must be planned so that they meet the management constraints such as deadlines, precedence, and priorities. This indicates the need for an evolution control system that takes into account the special characteristics of the evolution (maintenance) phase of the software life cycle process that account for up to 75% of the cost of the software systems [71].

The evolutionary prototyping model, where a prototype evolves via a number of versions to the final system is shown in Figure 2. Under this evolution model, developers start evolving the software system from its fundamental concepts, then keep modifying the system in an interactive way with the customer until the system reflects the customer's real needs. The importance of an evolution control system in such an interactive, exploratory system development model is even more obvious than for the waterfall model. In this model all kinds of changes are going on simultaneously, corrective changes to reflect the real customer requirements after reviewing the designer's interpretation of portions of the

developed requirements, adaptive changes to the rest of the customer's real needs, and perfective changes to the fundamental concepts already accepted by the customers. The interactions between these different activities, the coordination among related ones, propagating the effects of each of these changes to the rest of the developed modules, and keeping track of which component belongs to which system version are the main goals of our evolution control system.

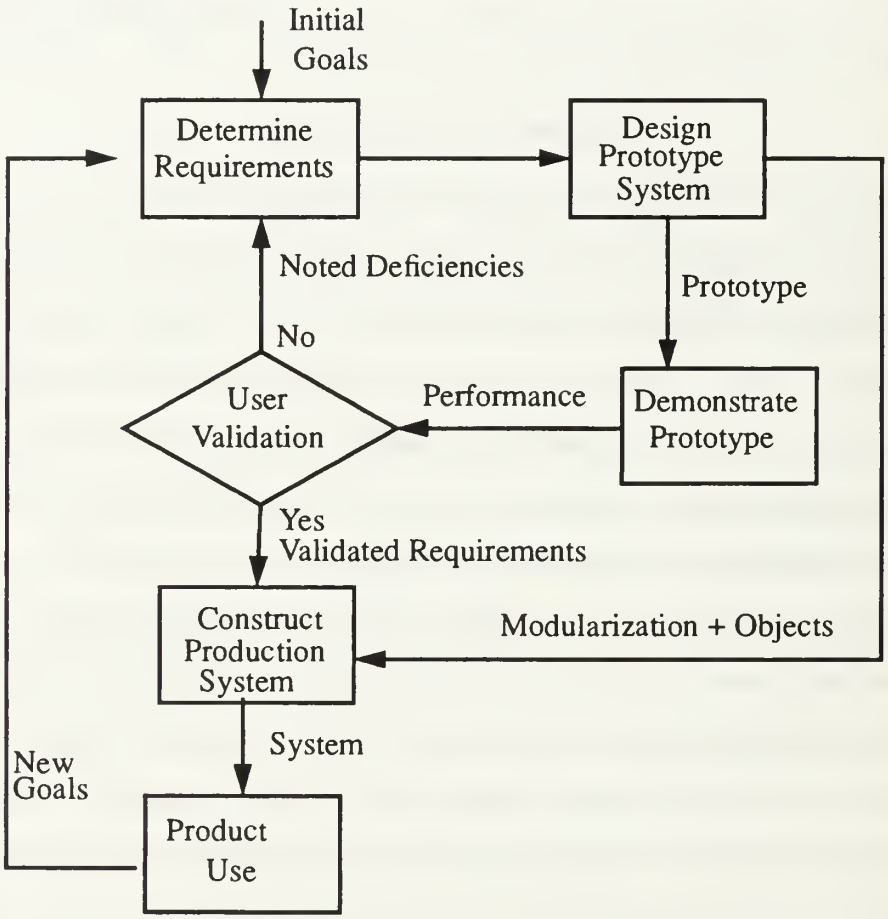


FIGURE 2. Rapid Prototyping Model

C. VERSION CONTROL AND CONFIGURATION MANAGEMENT

As indicated in [81], version control and configuration management is one of the fields in software engineering that has received much discussion and many proposals for proper version and configuration models in different domains, but little has been implemented,

and much remains to be done in developing techniques for ensuring the consistency of configurations and space efficient algorithms for version management.

According to [47] and [26], representations of the versioning process can be classified into two main models. The first model is the conventional Version Oriented Model (VOM) in which a system is divided into modules each of which is versioned independently from the other modules. To configure a system one has to select a version of each module of the system. This makes *version* a primary concept while *change* is a secondary concept as a difference between versions. Both SCCS and RCS [75] [76] [78] conform to this model. The second model is the Change Oriented Model (COM). In this model the functional change is the primary concept. Versions are identified by a characteristic set of functional changes. To configure a system in this model, one has to select a set of mutually compatible functional changes. Versions in this model are global, meaning that to examine a module one has to specify a single version of the system first, then proceed to the required module. The Aide~de~camp software management system [1] belongs to this model. On the other hand, in a VOM system, to examine a module one has to select the module first, then individually select which version of this module is the target.

Reference [26] also defines two more models: The composition model and the long transaction model. The composition model is a natural outgrowth of the VOM model. A configuration in this model consists of a system model and version selection rules. A system model lists all the components of a system. Version selection rules define which version is to be selected for each component to compose a configuration rather than allowing the user to manually pick component versions. Selection rules may be specific, i.e., repeated application of the selection rules will result in the same component versions (bound configuration). Otherwise the selection rules are generic (partially bound or configuration template), i.e., application of the rules at different times may result in a different bound configuration, e.g., choosing the latest version.

The long transaction model supports the evolution of whole systems as a sequence of apparently atomic changes, and coordinates the change of software systems by teams of developers. Developers work primarily with configurations rather than individual

components. A change is performed in a transaction. A specific configuration is selected as a starting point for changes which implicitly determines the version of the components. The modifications to this configuration are not visible outside the transaction until the transaction is committed. Multiple transactions are coordinated via concurrency control schemes to guarantee no loss of changes. The result of the committing of a transaction is a new system configuration version either on the same development path or branch from an existing development path resulting in a new alternative (variation) development path.

Our work utilizes concepts from both the VOM and long transaction models. Applying a top level evolution step to a base version of a software system leads to versioning of both the individual components involved in the change and the entire software system producing a new configuration version (version of a whole system). In addition our system automatically coordinates teamwork in such a way that concurrency control is done at a higher level of abstraction, i.e., the serialization of dependent evolution steps is done by serializing their assignment to developers in the same order and excluding the need for the traditional locking schemes. Including the evolution steps, with all the data they have about the change they implement, as nodes in the bipartite evolution history graph facilitates evolution history tracing.

The three main aspects of organizing software objects defined in [31]:

1. evolution: The software objects should be organized in such a way that makes it possible to view their evolution and origin,
2. membership: grouping software objects in a way that they are easy to find, and
3. composition: putting the appropriate components together for the composition of a new release,

are similar to the underlying concepts used by our mechanism; the difference is that we use composite objects to represent the membership organization and to define the composition organization. This same structure represents configurations of systems and their subsystems.

Our concept of composite entities and its generalization to fit system configurations is also similar to that used in PACT [70]. Our system uses a computed labeling function and a single versioning mechanism for automatically versioning individual objects as well as

configuring a system (as a composite object). Simplifying version control and configuration management and making it transparent to the user without requiring his/her intervention, as it is the case in our system, are two of the main goals of a good version control and configuration management system as set forth by Feldman in [27].

Our system takes care of planning, scheduling, status accounting and auditing of changes via explicit representations of steps as well as software component versions. Each step has a unique step number (generated automatically by the system) and is associated with all the relevant information such as dependent modules, affected modules, who made the changes and when, and the current status of the step in addition to a description of the motivation for these changes. This enables the system to answer questions similar to those mentioned in [49] such as: what changes were made in step #X, what components were affected by this change, what changes were made to the system after a certain date, and so on.

D. TOOL SUPPORT FOR CONFIGURATION MANAGEMENT AND COOPERATIVE DESIGN

According to Kaiser and Perry [37] the main tools that propagate changes among modules are listed below. However, none of these support the enforced model of cooperation among programmers necessary for large maintenance/evolution projects or automatically assign tasks to programmers:

Make: a UNIX tool that rebuilds the entire software system. It invokes the tools specified in the "Makefile" on changed files and their dependent files. Make is used for regenerating up-to-date executables after source objects have been changed.

Build: is an extension to make that permit various users to have different views of target software system. A "viewpath" defines a series of directories to be searched by make to locate the files listed in the makefile.

Cedar: the Cedar System Modeler uses an advanced version of the Make tool with version control to invoke the tools on a specific versions of files. This System informs the

"Release Master", a programmer, about any syntactic interface errors. The Release Master is responsible for making work arrangements with responsible programmers.

DSEE: the Apollo Domain Software Engineering Environment also uses a Make-like tool with version control. DSEE also has a monitoring facility that permits programmers and/or managers to request to be notified when certain modules are changed.

Masterscope: Interlisp's Masterscope tool maintains cross-reference information between program units automatically. It also approximates change analysis of potential interference between changes by answering queries about syntactic dependencies among program units.

SVCE: the Gandalf System Version Control Environment performs incremental consistency checking across the modules in its database and notifies the programmer of errors as soon as they occur. The consistency checking is limited to syntactic interface errors. It supports multiple programmers working in sequence but does not handle simultaneous changes.

Kaiser and Perry [37] [38] [65] also describe Infuse, a system that automates change management by enforcing programmer cooperation to maintain consistency among a sequence of scheduled source code changes. Infuse automatically partitions these modules into a hierarchy of experimental databases. This partitioning may be done according to the syntactic and/or semantic dependencies among the modules or according to project management decision. Each experimental database provides a forum for the programmers assigned to its modules or their managers, and provides also for consistency checking among those modules (meaning that the interface between the modules must be correct and that the modules can compile and link successfully). Consistency checking among the experimental database modules is a pre-condition for merging a database back to its parent experimental database. Infuse automatically partitions the database into experimental databases but programmers are assigned to the these databases manually.

In our system tasks and copies of the associated versions of software components are assigned automatically to designers (programmers) according to their dependencies. Versions are generated automatically as soon as the work is done. Syntactic and semantic

consistency checking for source code can be implemented by associating declarations of consistency constraints with steps, and triggering the required checking actions as part of the commit protocol.

E. APPROACHES TO SCHEDULING EVOLUTION STEPS

A scheduling problem in a real-time system is described by three basic concepts: the model of the system, the characteristics of the tasks to be scheduled, and the objective of the scheduling algorithm [67].

First, the system model in our case consists of a set of m designers $D = \{d_1, d_2, .., d_m\}$. Those designers are of three different expertise levels {low, medium, high}. The scheduling algorithm determines the order of the execution of tasks by each designer in such a way that resource, precedence, and timing constraints are met. In our system resources required by a task other than the designer resources are assumed to be available as soon as the task is assigned.

Second, the nature of a task, an evolution step in our case, is characterized by its timing constraints, precedence constraints, and resource requirements. The timing constraints of a task are generally defined in terms of one or more of the following parameters [67]:

1. The arrival time, T_a: The time at which a task arrives at the system.
2. The earliest start time, T_{est}: The earliest time at which a task can start execution. (invariant: $T_{est} \geq T_a$).
3. The worst case execution time, T_c: The execution time of a task is always less than this time.
4. The deadline, T_d: The time by which a task must be completed.
   The following invariant is always true: $0 \leq T_a \leq T_{est} \leq T_d - T_c$

While all the tasks and their timing constraints are known beforehand in a static system, tasks arrive at arbitrary times in a dynamic system, so that the number of tasks to be scheduled as well as their arrival times are unpredictable.

In many conventional real-time systems a fixed ***priority*** is assigned to each task to reflect the criticalness of the deadlines, and tasks are executed in an order determined by their priorities. These priorities are adjusted (manually), during the testing period, until the system designer is convinced that the system works. This approach works only for

relatively simple systems, because of the difficulty of determining a good priority assignment for a system with a large number of tasks by such a test-and-adjust method. Also, once the priorities are fixed on a system, it is very expensive to modify the priority assignment [67]. Often, priorities are assigned to tasks based only on their importance, without a complete analysis of how these priority assignments will affect the timing characteristics of other tasks. Using priorities in this way make it more difficult to satisfy timing constraints of all the tasks [83].

The relations between the tasks are determined by the ***precedence*** constraints among these tasks. If a task T_i must be completed before another task T_j can be started then we say T_i precedes T_j. The precedence graph of a set of tasks is a directed acyclic graph. This precedence graph is known in advance in static systems. In dynamic systems where new sets of interrelated tasks arrive arbitrarily, the precedence graph is known only when the task set arrives.

Third, the objective of an algorithm for scheduling a set of tasks is to determine whether there exists a schedule for executing the tasks that satisfies the timing, precedence, and resource constraints, and to calculate such a schedule if one exists.

Task scheduling in real-time systems can be static or dynamic. A static approach performs the calculation of the schedules for tasks off-line. It requires prior knowledge of the characteristics of the tasks. On the other hand, a dynamic approach calculates schedules for tasks "on the fly". Despite the fact that static approaches have low run-time cost, they are inflexible and cannot respond to a changing environment with unpredictable behavior. This inflexibility leads to calculating the schedule for the whole system when a new tasks are added, which is expensive in terms of both time and cost. In contrast, dynamic approaches involve higher run-time costs, but they are flexible to adapt to environment changes. A survey of static and dynamic scheduling approaches can be found in [67].

Task scheduling can also be characterized as preemptive and nonpreemptive. A task is preemptive if its execution can be interrupted by other tasks and resumed afterwards. A task is nonpreemptive if it must run to completion once it starts.

1. Scheduling Tasks with Precedence Constraints

Scheduling tasks with arbitrary precedence constraints and unit computation time in multiprocessor systems is NP-hard for both the preemptive and nonpreemptive cases [67] [84]. Scheduling nonpreemptive tasks with arbitrary ready times is NP-hard in both multiprocessor and uniprocessor systems [67] [83] which excludes the possibility of the existence of a polynomial time algorithm for solving the problem. Hong and Leung [34] proved that there is no optimal on-line scheduler can exist for task systems that have two or more distinct deadlines when scheduled on m identical processors where $m > 1$.

Scheduling evolution steps to more than one designer with arbitrary precedence constraints and arbitrary deadlines is the same problem as that of multiprocessor scheduling mentioned above which is shown by many researchers to be NP-hard. These negative results dictate the need for heuristic approaches to solve scheduling problems in such systems. In the rest of this section we review some of the relevant task scheduling heuristics used in similar problems and highlight their relevance to our work.

In [72] Stankovic et al. present an $O(n^2)$ heuristic scheduling algorithm for scheduling a set of independent processes on a set of identical processors. A task (process) in this model is characterized by an arrival time T_A, a deadline T_D, a worst case computation time T_C, and a set of resource requirements $\{T_R\}$. Tasks are independent, non periodic and non-preemptive. The authors stated that scheduling a set of tasks to find a full feasible schedule is a search problem with a search tree as the search space. The scheduling algorithm starts at the root of the tree which is an empty schedule. It tries to extend the schedule by moving to the one of the nodes in the next level of the search tree until it reaches a full feasible schedule. It is worth noting that, during the expansion of the schedule, an intermediate node is a partial schedule, while leaf nodes (terminal node) represent full schedules. It is clear that not every terminal node corresponds to a feasible schedule. To extend the schedule to a node of the next level of the search tree, the algorithm uses a boolean function called "strongly-feasible" to determine if the partial schedule is strongly-feasible or not. A partial schedule is strongly-feasible if all schedules reached by extending it by each of the remaining tasks are also feasible. This means that if a partial

feasible schedule is found not to be strongly-feasible because a task T misses its deadline, then the search should stop on this path since none of the future extensions of task T will meet its deadline. However, it is possible to backtrack to continue the search in such cases. After deciding that a partial schedule is strongly-feasible, a heuristic function (H) is used to direct the search to a plausible path.

This algorithm works as follows: Given a particular heuristic function H, the algorithm begins with an empty partial schedule. Every step of the algorithm includes (a) determining if the current partial schedule is strongly-feasible, and if so (b) extending the current partial schedule by one task. This task is selected by applying the H function to all the tasks remaining to be scheduled and determining the one with the minimum H value.

Some of the H functions used in [72] are Minimum deadline first(Min_D), Minimum processing time first (Min_P), Minimum earliest start time first (Min_S), Minimum laxity first (Min_L), and the combinations (Min_D + Min_P) and (Min_D + Min_S).

In [67], Ramamritham et al. introduce an O(nk) version of the algorithm introduced in [72] by considering only k tasks of the remaining tasks to be scheduled for applying the H function and evaluating the strongly-feasible function.

Both [72] and [67] use a vector data structure for each type of resources to maintain the earliest available time for each resource of each type. In our algorithm for scheduling evolution steps we extend this algorithm to handle the case where there are precedence constraints between pairs of steps, and keep a vector of earliest available times of designers for each expertise level. Details of our algorithm are in Chapter IV.C.

In [82], Xu and Parnas present a pre-run time (static) algorithm to find a feasible schedule if one exists on a single processor for a set of processes with arbitrary precedence and exclusion relations and arbitrary deadlines. This algorithm assumes that release times, deadlines, precedence and exclusion relations are known in advance.

In [84], Xu and Parnas extend their pre-run time algorithm presented in [82] above to find a feasible nonpreemptive schedule whenever one exists on M identical processors for the same set of processes defined above. In both cases the algorithms use a

branch and bound technique that has a search tree where at its root node they use an earliest start time first strategy to compute a schedule called the "valid initial solution" that satisfies the release time constraints and all of the initial precedence and exclusion relations. For each node in the search tree a lower bound on the lateness of any schedule leading from that node is computed. The algorithm branches from the node that has the least lower bound among all unexpanded nodes. This operation continues until either a feasible solution is found or there exist no unexpanded node that has a lower bound less than the least lateness of all valid initial solutions found so far. This algorithm requires all the constraints to be known in advance which is not the case in our problem. It also does not provide any response to changing any of the constraints of the task set or the arrival of new tasks.

III. REQUIREMENTS ANALYSIS

A. GOALS AND JUSTIFICATIONS

The main goal of this dissertation is to develop a model and algorithm for an evolution control system (ECS) for the software evolution/prototyping process. This system provides automated support for changes in plan during the execution of the plan and automatic decision support for planning and team coordination based on design dependencies captured in a configuration model. This ECS is needed to control and coordinate the overwhelming changes dictated by the evolutionary nature of the software prototyping/ development process. These prototypes experience iterative and exploratory modifications through large numbers of versions and variations to cope with customers' changing and growing needs. Coordination is needed to avoid rollbacks, redundancy and inconsistency. Control is a necessity for managing both the design data (version/configuration control) and the design team via orchestrating task assignment to support management policies. This allows the software development team to concentrate on what is needed to fix or improve system components rather than worrying about managing this enormous amount of data. The following are the main goals for our proposed system:

1. Manage the evolution steps from the moment a system is proposed until its completion.
2. Reach a feasible schedule that meets the deadline requirements of all the active steps or automatically cancel the lowest priority deadlines until a feasible schedule that meets the deadlines of the higher priority steps is reached.
3. Support teamwork by identifying the steps that can be scheduled concurrently.
4. Support incremental replanning as additional information becomes available.
5. Minimize wasted design effort due to reorganization of the schedule as well as workers forced to wait for completion of sub-tasks via the immediate detection of new dependencies forced by this reorganization and the suspension and rescheduling of any of the affected assigned-steps.
6. Ensure system integrity via propagating change consequences (induced steps) to maintain the global consistency of the design database and provide serializability of updates.
7. Provide automated version control and configuration management.
8. Efficient use of space and time for the design database and scheduling algorithm.

The first goal is needed to help the prototype manager control the large number of changes dictated by both the exploratory nature of the prototyping process and customer feedback, and keep track of which designer is performing which change. The second goal is crucial for planning to accomplish the required changes. The third and fourth goals are to cope with the dynamics of the prototyping process where the steps to be scheduled are only partially known. Time required, the set of sub-tasks for each step, and the input/output constraints between steps are all uncertain and subject to change as evolution steps are carried out. The global consistency of the design database is covered by the fifth goal. The sixth goal seeks to automate the version control and configuration management in this dynamic environment to save designers' time and effort. They need not worry about managing the complicated design database and may concentrate on their main task of performing the required tasks. The last goal is an implementation requirement for saving storage space, especially in this exploratory environment where many alternatives are explored that require much storage space, and to find a time-efficient scheduling algorithm that does not impact the timing constraints of the scheduled steps.

B. GRAPH MODEL OF SOFTWARE EVOLUTION

Since the main purpose of the ECS is managing software evolution in a rapidly evolving system, we review a graph model of software evolution that constitutes the context for building the ECS [52] [58]. The goal of this model is to provide a framework for integrating software evolution activities with configuration control [52]. The model of software evolution has two main elements: system components and evolution steps. System components are immutable versions of software source objects that cannot be reconstructed automatically. Evolution steps are changes to system components that have the following properties in the original version of the graph model [52]:

1. A top-level evolution step represents the activities of initiation, analysis, and implementation of one change request.
2. An evolution step may be either atomic or composite.
3. An atomic step produces at most one new version of a system component. This property is no longer true in our model in order to include the cases in which an atomic step is applied to an originally atomic component that needs to be decomposed according to

some design considerations. This decomposition may lead to the production of more than one component.This modification is illustrated in section C.2.e later in this chapter.

4. The inputs and outputs of a composite step correspond to the inputs and outputs of its substeps.
5. The model allows steps that do not lead to the production of new configurations, e.g. design alternatives that were explored but not included in the configuration repository.
6. Completely automatic transformations are not considered to be steps and are not considered in this model.
7. The graph model can cover multiple systems which share components, alternative variations of a single system, and a series of configurations representing the evolution history of each alternative variation of a system.
8. A scope is associated with each evolution step which identifies the set of systems and variations to be affected by the step. The scope is used to determine which induced evolution steps are implied by a change request.

The evolution history is modeled as a graph G=[C, S, CE, SE, I, O]. This graph is a directed acyclic graph (bipartite with respect to the edges I and O). C and S are the two kinds of nodes (C: software component nodes, and S: evolution step nodes respectively). Each node has a unique identifier. C and S nodes alternate in each path that has only I and O edges. This represents the evolution history view of the graph. The edges represent the "part_of" (between a sub-component of a composite component and the composite component) and "used_by" relations (defined between components to represent the situation where the semantics or implementation of one component A depends on another component B; B used_by A) between the software components of a given configuration ($CE \subseteq C \times C$), the "part_of" relation between a substep of a composite step and the composite step ($SE \subseteq S \times S$), the input relation between the system components which must be examined to produce output components that are consistent with the rest of the system and the corresponding evolution steps($I \subseteq C \times S$), and output relation between evolution steps and the components they produce ($O \subseteq S \times C$). System components are immutable versions of software source objects that cannot be reconstructed automatically.

An "edge_type" attribute is used to distinguish between the two kinds of edges representing the relations "used_by" and "part_of" defined on the set of edges $CE \subseteq C \times C$. The "used_by" relation can be used for automatic identification of inputs of

proposed evolution steps and identification of the induced steps triggered by a proposed step. A review of the formal definitions of some of the concepts mentioned above, as defined in the original graph model [52], is introduced below. Some modifications to some of these definitions and the reasons for them are indicated where it takes place.

a) the set of input components of a step: input (s: S)= {c: C| [c, s] ∈ I} (1)

b) the set of output components of a step: output (s: S)= {c: C| [s, c] ∈ O} (2)

c) Atomic step: atomic (s: S)= ~ EXISTS (s1: S:: s1 part_of s) (3)

d) one component affects another if both components are identical or if the first is used in the derivation of the second:

ALL(c1, c2:C:: c1 affects c2 ⇔ c1 used_by* c2) (4)

where "used_by*" is the reflexive transitive closure of the "used_by" relation defined above.

e) The output of a composite step includes all the outputs of its sub-steps:

ALL(s1, s2:S, c: C:: s1 part_of s2 & c ∈ output (s1) ⇒ c ∈ output (s2)) (5)

f) Every input to a sub-step either must be affected by some input to the parent step, or must affect some output of the sub-step

ALL(s1, s2: S, c: C:: s1 part_of s2 & c1 ∈ input (s1) ⇒

EXISTS (c2: C:: c2 ∈ input (s2) & c2 affects c1) |

c2 ∈ output (s1) & c1 used_by c2))) (6)

g) The primary input concept can be formalized by introducing the attributes object_id, version_id and variation_id that apply to versions to yield a unique identifier for the object and variation associated with each version. Variations represent alternative choices, which may correspond to different formulations of the requirements in the context of prototyping, or different kinds of system software (operating system, window manager, etc.) in the context of product releases. Versions represent the evolution history of a

particular variation. An input to a step is primary if and only if it is the previous version of the same object and belongs to the same variation as the output of the step. Same variation and primary input concepts are defined as follows:

ALL(c1, c2:C:: c1 same-variation c2 <=> object-id(c1) = object-id (c2) &

variation-id(c1) = variation-id(c2)) (7)

ALL (s: S, c1: C:: c1 primary_input s <=>

c1 ∈ input (s) & EXISTS (c2: C:: c2 ∈ output (s) &c1 same-variation c2) (8)

The above definition does not consider the inputs that leads to a new version of an object on a different variation (split) as primary inputs. This is the reason we define an input to a step to be a primary input if and only if it is the previous version of the same object as the output of the step. This concept can be formalized as follows.

ALL (s: S, c1: C:: c1 primary_input s <=>

c1 ∈ input (s) & EXISTS (c2: C:: c2 ∈ output (s) & object_id (c2) = object_id (c1)

& version_id (c2) = version_id (c1) + 1)) (9)

h) The scope of a top-level step consists of the components affected by its inputs.

scope(s:S) = {c1:C | EXISTS (c2:C:: c2 ∈ input (top(s)) & c2 affects c1)} (10)

i) The set of induced steps are defined as follows.

induced-steps (s1) ={s2:S | EXISTS (c1, c2:C:: c1 primary_input s1

& c2 primary_input s2 & c1 affects c2 & current(c2) & c2 ∈ scope (s1))} (11)

where a component is current if there is no later version of the same variation of the same object:

ALL(c1:C:: current (c1) ⇔ ¬ EXISTS(c2:C:: c1 D^+ c2 & c1 same_variation c2))

(12)

where D^+ is the transitive closure of the relation $D = (I \cup O)$ (13)

1. Summary of Modifications to the Original Graph Model

In the previous section we have introduced three basic modifications to the original graph model [52]. First, two sets of edges are added to the original graph model to represent the "part_of" relation (between a sub-component of a composite component and the composite component) and "used_by" relation (defined between components to represent the situation where the semantics or implementation of one component A depends on another component B; B used_by A) between the components of a given configuration ($CE \subseteq C \times C$), and the "part_of" relation between a substep of a composite step and the composite step ($SE \subseteq S \times S$).

The second modification is relaxing the restriction on an atomic step to produce at most one output component. This modification is needed to account for the cases where a designer assigned an atomic step (atomic steps are always parts of a top level evolution step that is used for control purposes and is not assigned to a designer. See section III.B.7 for details) needs to decompose the assigned module which may lead to the production of more than one output from the step. Since the ECS has control over the components in the design database, not over each designer's workspace, the designer who does the decomposition should commit these new modules to the design database where the ECS can propose a step for each new incomplete subcomponent as parts of the top level evolution step. The manager reviews the proposed substeps, add the management constraints, approve these substeps, and then the system automatically schedules these substeps to the rest of the design team, supporting teamwork.

The third modification is changing the primary input concept to be compatible with the version control and configuration model defined below. An input to a step is primary if and only if it is the previous version of the same object as the output of the step, whether the output version is on the same variation as the input of the step or splitting a new variation. This modification makes some object versions belong to one or more variations to help trace the evolution history of each variation to the initial version of each object.

a. Version and Variation Numbering

As soon as the input base version of a step is bound, the system assigns the version and variation number of the output object for the step. The variations are assigned successive numbers beginning with 1 for the initial variation. Versions along each variation are assigned successive numbers starting with 1 at the root version of the initial variation. This means that the new version number is the base version number plus one, while the variation number has two possibilities: the first possibility is to keep the base version's variation number at the time the step is assigned. This occurs when the base version is the most recent version on its variation line at the time the step is assigned. The other possibility is to use the "next" variation number, which is the highest variation number plus one. This labeling function is the same for both atomic or composite objects (the entire software system is represented as a composite object).

Let V_base1, V_base2 be different versions of an object to which an evolution step is applied, and V_new be the output version produced by the step. Let S be the primary input set of the step, then the version and variation numbers of the output version of the step are calculated as follows:

```
Case size(S) is:
    when 0 =>
        -- newly created object starts as version 1 on variation 1 of that object.
        version_number (V_new) = 1
        variation_number (V_new) = 1
    when 1 =>
        -- normal case
        version_number (V_new) = version_number (V_base1) + 1
        if successor (V_base1) = none then
            variation_number (V_new) = variation_number (V_base)
        else
            variation_number (V_new) = highest_variation (object (V_base)) + 1
```

```
            end if
        when >1 =>
                -- merge case
                if version_number (V_base1) >= version_number (V_base2)
                then version_number (V_new) = version_number (V_base1) + 1
                    v = V_base1
                else version_number (V_new) = version_number (V_base2) + 1
                    v = V_base2
                end if
                if successor (v) = none
                then variation_number (V_new) = variation_number (v)
                else variation_number (V_new) = highest_variation (object (v)) + 1
        end if
```

where highest_variation is a function that returns the highest variation exist for an object.

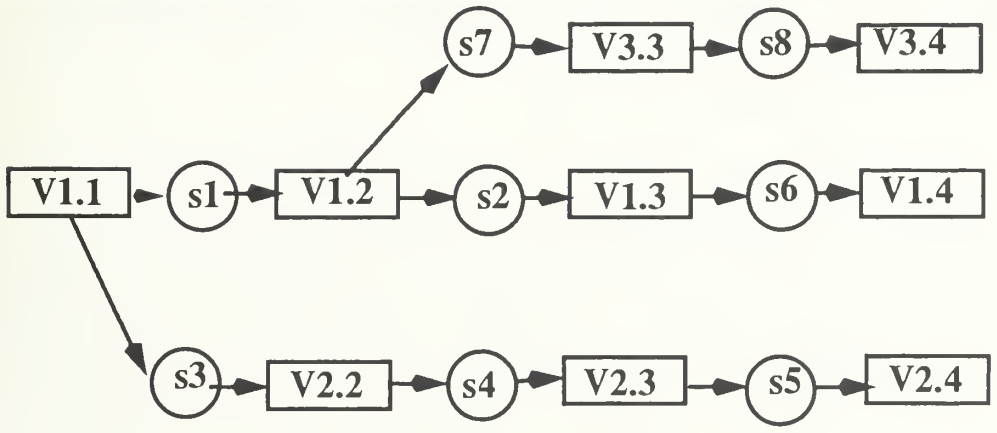


FIGURE 3. Variation and version numbering (case 1)

Cases 0 and 1, of this numbering mechanism are illustrated by an object evolution graph in Figure 3. V1.1 corresponds to a newly created object, V1.2, V1.3 and V1.4 are examples of the creation of versions on the same variation. V2.2 represent the first split, and V3.3 represents the second split creating the new variations 2 and 3 respectively.

Case >1, is included to make this numbering system compatible with the ongoing work for automated version merging capabilities under parallel development [22], [23]. This case is also illustrated by an object evolution graph in Figure 4. V1.5 is the result of merging V1.4 and V2.4. while V3.5 is the result of merging V1.4 and V2.3.

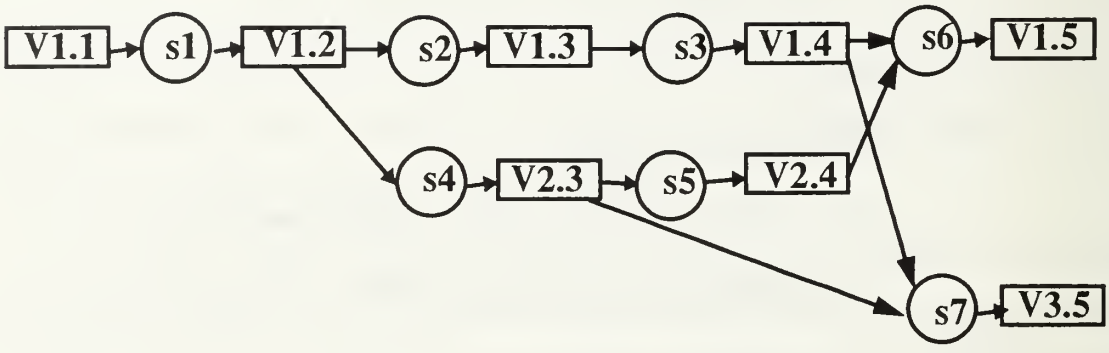


FIGURE 4. Variation and version numbering in case of merge (case >1)

This labeling function allows a version to belong to more than one variation which is a necessary modification to [52] to simplify the process of tracing the development history of a version and to keep a logical and realistic development history.

b. Configuration Management

As mentioned in Section 3 above, the configurations of the different software systems/prototypes are represented by a hierarchical structure according to the levels of the decomposition of each system. This hierarchical structure is a directed acyclic graph with its nodes representing the different components of the system and the edges representing the relations among these components which are "part-of" and "used_by". A top level evolution step producing a new version of one or more of the system components eventually leads to producing a new version of the base configuration. This is done by propagating the versioning process of the modified components up the hierarchy from the

levels of the newly created versions to their parents and all the way up to the root of the tree which represents the system under evolution. It is worth noting that the parent of a modified component should have a new version because of the change to its children list since this list should point to the new modified child version. This is formalized as follows:

For i= N down to 2

exists (new-version (component (level(i)))) &

component (level(i)) part_of component (level (i-1)) &

¬exists (new-version (component (i-1)) → new-version (component (i-1))

where N is the number of levels in the configuration graph in which the root is level number 1. This rule is limited to the scope of the step which is the base configuration.

Figure 5 shows an example for building a new configuration as a result of committing a top level evolution step. This example assumes the use of formal specification in software evolution that limits the impact of evolution steps (i.e., the implementation of the composite modules uses its own specification and the specification of its sub-modules, but not the implementation of the sub-modules). In this example, the primary input of the step S1 is Oe.spec.1.1 (the specification of the module Oe) that affects its implementation and the implementation of its parent module Oe.imp.1.1 and Mb.imp.1.1 respectively. A substep is created for each changing component, S1.1, S1.2, and S1.3 for the components Oe.spec.1.1, Oe.imp.1.1, and Mb.imp.1.1.

Committing these steps after the modifications are done will trigger the following actions: 1) each substep produces the next version on the same variation line of its primary input component (assuming no splits), this means the components Oe.spec.1.2, Oe.imp.1.2, and Mb.imp.1.2 are produced as outputs to the corresponding steps. 2) As a result of committing the three substeps, the commitment of the top level step S1 is triggered. Committing S1 means performing a Depth First Search (DFS) on its input configuration Sys 1.2 looking for the primary input components of the step, from the leaf

nodes and up, if a parent of any of the primary input component does not have a new version, then a new version is created for it and its composition list is updated to point to

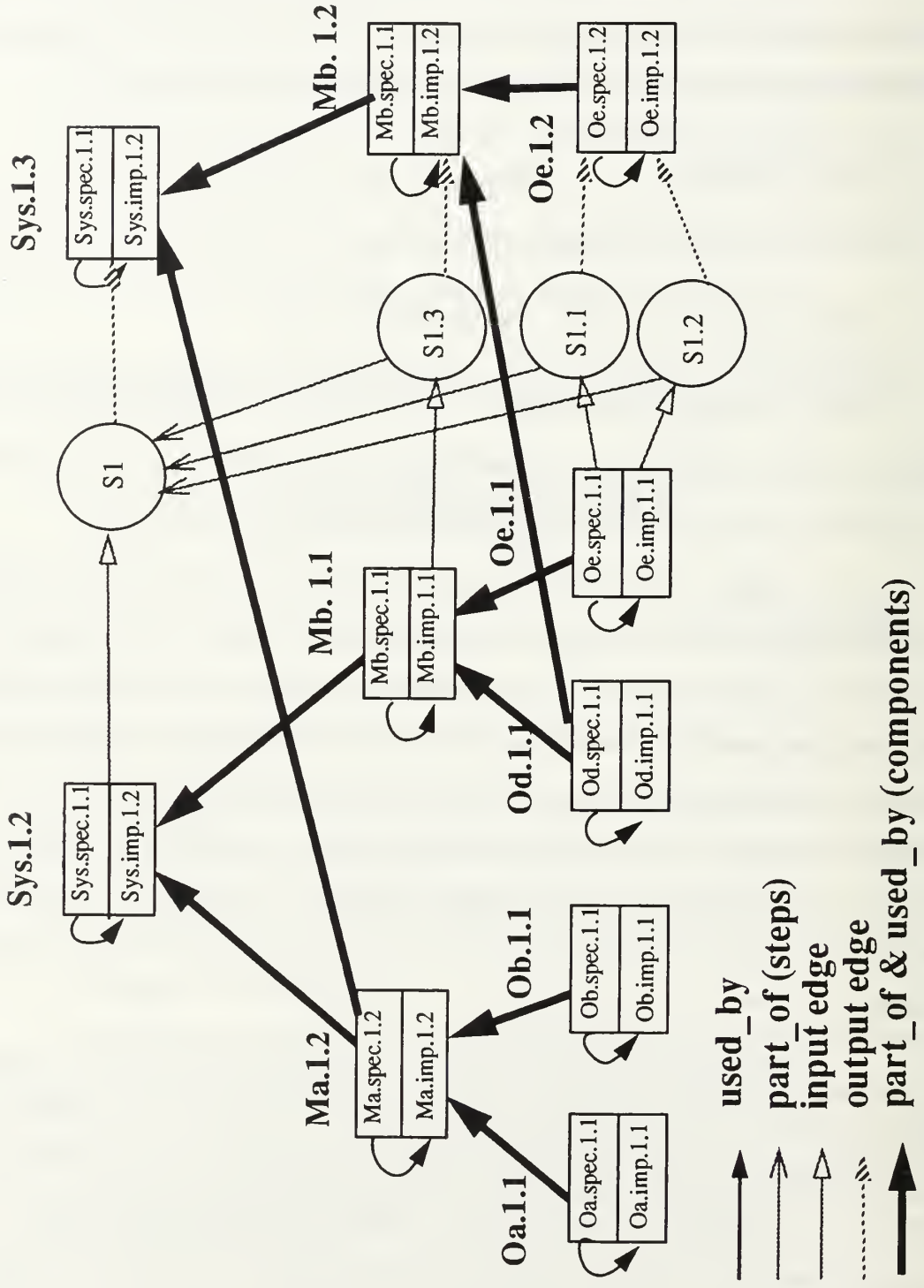


FIGURE 5. Building a new configuration

the new version of its modified child. 3) This versioning of the parents is recursively done for all the parents of the newly created versions in the previous step until the root node is versioned creating the new configuration Sys.1.3. The reason for performing DFS on the primary configuration instead of just building the configuration from the deepest versioned component all the way up is that each component may belong to more than one configuration while we need only a new version for a specific one of them (the primary input configuration).

2. States of Evolution Steps

The dynamics of the evolution steps are modeled by associating six different states with each step to express the different activities each step has to undergo during its lifetime. The state transition diagram in Figure 6 shows the different explicit decisions that have to be made by the management to cause the transition from one state to the other. It also shows the automated transitions from the scheduled state to the assigned state and vice versa (explained in detail in subsections c, and d below). By controlling the states of the evolution steps, the evolution manager exercises direct control over both software evolution/development and the resulting software configurations. The following are the definitions of those states and the corresponding actions that cause the transition from one state to the other. These states are similar to those presented in [52] except that a new state called "assigned" has been added for the reasons explained below.

a. Proposed State

In this state a proposed evolution step is subjected to both cost and benefit analysis. This analysis also includes identifying the software objects comprising the input set of the step. An evolution step is created when it is proposed, which means that the initial status of a newly created step is "proposed". The create_step command takes as input a primary_input component and a base version of the whole system configuration and returns a unique step number, a set of components affected by the change to the primary_input component and a set of secondary input components (the components used_by the primary_input component). A "proposed" step is generally added to the configuration graph as an isolated step node that does not have any input, output or part_of edges (except

when an old version is used that has existing specific reference). This is because the primary and secondary input attributes are mostly generic inputs (object_id and variation_id only).

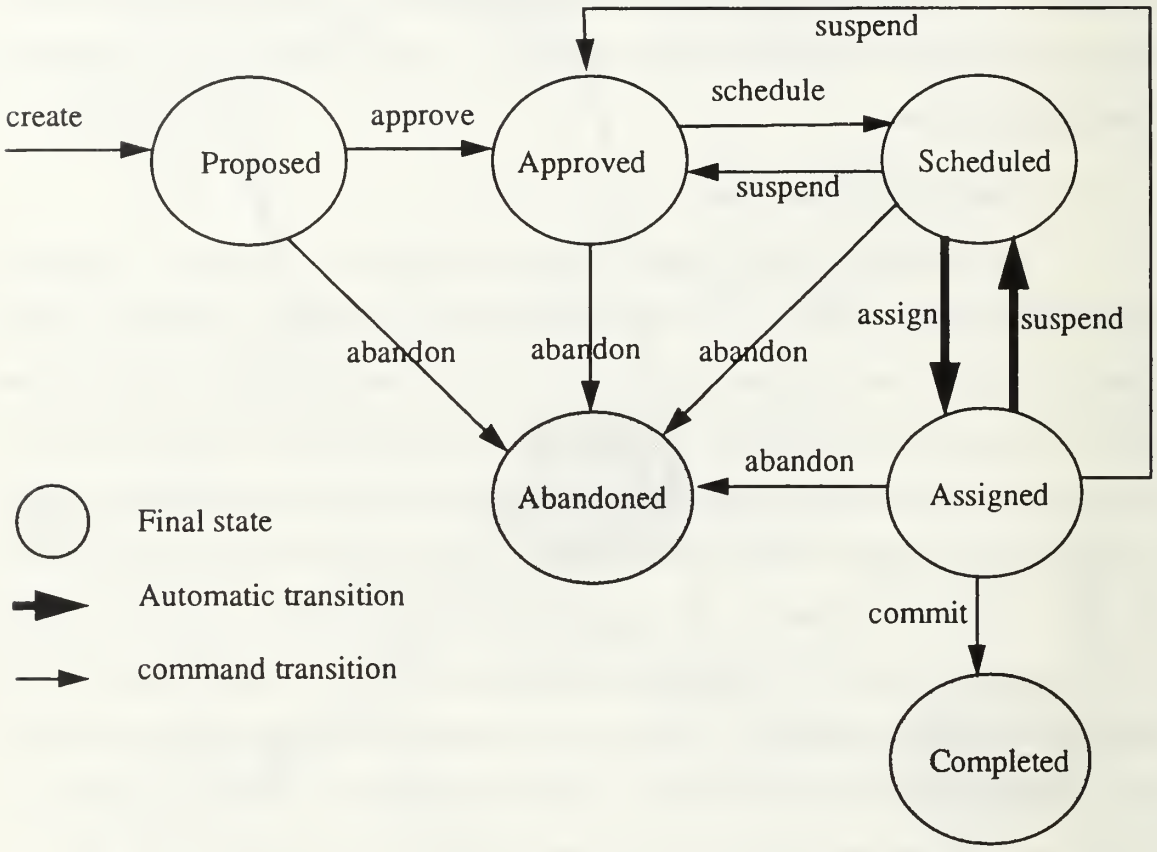


FIGURE 6. Evolution step's state transition diagram

Inputs to an evolution step can be specified by references to either a generic object or a specific object version. Generic object references are commonly used to denote the current version of the object. A generic object reference consists of an identifier for the object and an identifier for a variation of that object [52]. Generic object references can be used only while the step is in the "proposed", "approved" or "scheduled" states, and must be bound to specific versions before the step can enter the "assigned" state. If the binding is not specified manually, it defaults to the version from the previous version of the entire prototype with respect to the serialization order of the top level evolution step. The version

bindings of a composite step are inherited by its sub-steps to ensure consistency. Specific object references are usually used to define inputs to steps in cases when an older version is used, which often coincide with the creation of new variations.

If a specific reference is to be used, it has to be entered into the system by the software manager using the edit_step command before issuing the schedule_step command. This is because it is used for the calculation of the serialization order among the steps to be scheduled.

b. Approved State

In this state the implementation of the step has been approved but not scheduled yet and the input set of the step is not bound to particular versions. Approval of a proposed step by the management and the Change Control Board advances its status to "approved", and triggers the decomposition process to create an atomic sub-step for each primary or affected component of the step. These sub-steps inherit the status of their super-step which is "approved" in this case, and are added to the configuration graph with a part_of edge between each sub-step and its super-step.

It is also in this state that the substeps are augmented with attributes that include the estimated duration of each sub-step and management scheduling constraints such as precedence, deadline, and priority. The "approved" state can also be reached from both the "scheduled" and "assigned" states using the "suspend_step" command to suspend work on a step due to budget cuts or other management reasons.

c. Scheduled State

In this state the implementation has been scheduled and the step is not yet assigned to a designer. When a designer is available the step is assigned to him/her and its status is automatically advanced to "assigned". The "scheduled" state is reached from the "approved" state via the command "schedule_step" that indicates that the management constraints are complete and enables the scheduling and job assignment mechanisms. The scheduling mechanism produces an updated schedule containing the newly scheduled step. A schedule specifies the expected starting and completion times for the step. The scheduled state can also be reached from the assigned state when a step is suspended by the system

because of a new dependency imposed due to adding a new step to the schedule or because of modified scheduling constraints due to editing an existing step. This leads to rescheduling of the suspended step.

d. Assigned State

In this state the step is assigned to the scheduled designer, all inputs are bound to particular versions, and unique identifiers have been assigned to its output components, but these components are not yet part of the evolution history graph. A composite step enters the assigned state whenever any of its substeps is assigned.

The assigned state is reached automatically from the scheduled state. When a designer is available, the schedule is used to determine his/her next assignment. If his/her next assignment is ready to be carried out then the step status is automatically advanced to "assigned" and the designer is informed of the new assignment. When a step is assigned, the version bindings of its inputs are automatically changed from generic to specific. An edge is added as an input edge between the primary input component of the step and the step itself in the configuration graph. It is in this state where the designer gets his/her assignment, does the required modification/development, and integrates and tests these modifications before the transition to the final state "completed".

e. Completed State

In this state the outputs of the step have been verified, integrated, and approved for release. This is the final state for each successfully completed step. This state can only be reached from the assigned state using the "commit_step" command. In this state the output components of the step have been added to the configuration graph. An output edge has also been added to the configuration graph between the step and its output component(s). A composite step enters the completed state when all of its substeps are completed

f. Abandoned State

In this state the step has been cancelled before it has been completed. The outputs of the step do not appear as components in the evolution history graph. All

completed work, if any, of the step and the reasons why the step is abandoned are stored as attributes of the step for future reference. This is the final state for all steps that were not approved by the management or cancelled in the "approved", "scheduled" or "assigned" states.

g. Relation to the Original Graph Model

The Evolution Control System, ECS, uses the above states to represent the different actions an evolution step goes through during its lifetime. The difference from [52] is that an assigned state is added between the scheduled state and the completed state to differentiate between the cases in which a step is scheduled (planned) but not yet assigned to a designer and those cases where the step is assigned to a designer and the work is in progress. It is always desirable to have a complete schedule for planning purposes to determine the feasibility of accomplishing the required changes by certain deadline and meeting various management constraints. This differentiation is needed for our automated management system (ECS) in case of suspending or abandoning a step. If the step status is scheduled, then the response is removing it from the schedule, changing its status to approved or abandoned respectively, and updating the schedule. If the step status is "assigned", then before issuing the previously mentioned actions, a warning to the manager that an effort is about to be wasted is issued, and in case of his confirmation, "all completed work if any" has to be saved as an attribute of the step for future reference.

3. Constraints on State Transitions

The following constraints are imposed on some state transitions of composite steps and their sub-steps in order to ensure consistency in the evolution histories containing both composite and atomic steps. These constraints are the same as those stated in [52] with a slight modification to correspond to the newly added state:

1. When a step changes from the "approved" state to the "scheduled" state all of its sub-steps automatically make this transition.
2. When a step is rolled-back from the "scheduled" or "assigned" state to the "approved" state all of its sub-steps automatically make the same transition.
3. A composite step automatically changes from the "assigned" state to the "completed" state when all of its sub-steps have done so.

4. A composite step automatically changes to the "abandoned" state when all of its sub-steps have done so.
5. When a step changes to the "abandoned" state all of its sub-steps automatically make the same transition.
6. When a new sub-step is created, it enters the same state as its parent step and inherits all version bindings associated with the parent step.

4. Specifying Inputs to the Evolution Steps

Evolution steps have two kinds of inputs: primary inputs and secondary (non-primary) inputs. An input to a step is primary if and only if it is the previous version of the same object as the output of the step as defined in equation (9). An initial approximation of the secondary inputs of an evolution step can be derived from the "used_by" relation, since a step can depend on all the components used by its primary input. This is formalized as follows:

$$\text{ALL } (c1\ c2: C, s: S:: c2 \text{ used_by } c1 \,\&\, c1 \in \text{primary_input}(s) \Rightarrow c2 \in \text{initial_secondary_input}(s)) \quad (14)$$

The set of secondary inputs to a step should include all the component versions used by the output of the step. The above is a mechanically derived initial approximation of the set of secondary inputs. This approximation may need some manual adjustment, since design changes may introduce dependencies that did not exist in the previous version, and can remove some dependencies that did exist. Such adjustments are made via the editing commands provided by the ECS.

5. Induced Evolution Steps

When a step modifies a component, it implicitly induces a set of other steps to carry out the corresponding changes in all of the other components which depend on the one that is modified by the original step (the inducing step). This means that induced steps produce versions of their primary inputs which are consistent with the output version of the inducing step and in the scope of the current top level evolution step. This concept is formalized in equations (8, 9, 10), which provides the basis for automatic construction of the set of induced steps for a given inducing step.

The purpose of this construction is to alert the software engineers and the management of the impact of the proposed changes and to prevent consistency problems due to incomplete propagation of the consequences of a change.

Since the inputs to the top-level evolution step are bound to specific versions at the time the step is assigned, the set of induced steps cannot be influenced by any changes due to parts of any other top-level steps that may be executed concurrently. The predicate "current" is evaluated with respect to the version bindings of the top level step. For all but the initial version of the components, the "used_by" relationships can be derived from the secondary input relationship in the evolution history graph.

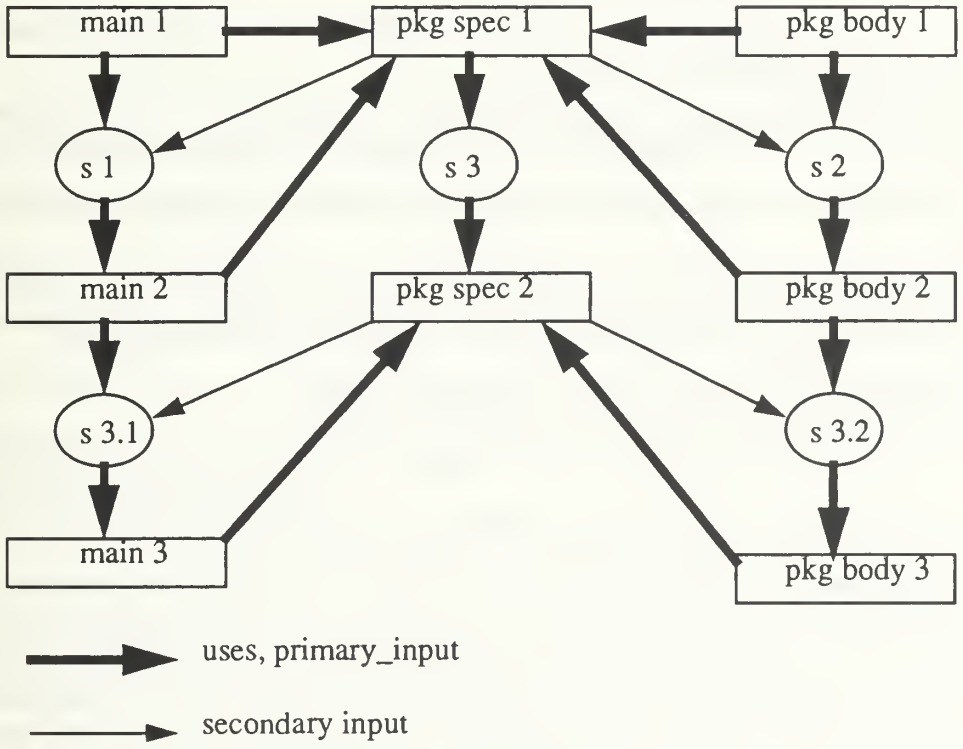


FIGURE 7. Induced evolution steps.

An example of induced steps in a small system implemented in Ada is shown in Figure 7 (taken from [52] p. 926). The initial configuration of the system shown in the figure consists of three components. The step s1 changes the main program without affecting the package specification and does not trigger any induced steps. Similarly the step s2 changes the package body without affecting the package specification or triggering any induced steps. The step s3 changes the package specification and triggers induced steps

s3.1 and s3.2, which must update the main program and the package body to be consistent with the new package specification.

6. Induced State Transitions

The following constraints are imposed on some state transitions of induced steps and their inducing step in order to ensure the consistency of the configuration. These constraints are similar to those stated in [52] with minor changes to reflect the addition of the "assigned" state to the model.

a. An inducing step can change from the "assigned" state to the "completed" state only when all of its induced steps have done so.
b. An induced step changes automatically from the "approved" state to the "scheduled" state when its inducing step does so.
c. When an inducing step is rolled-back from the "scheduled" or "assigned" state to the "approved" state all of its induced steps automatically make the same transition.
d. A "roll back" of an induced step can be done only by rolling back all of its inducing steps.
e. Abandoning an inducing step causes all of its induced steps to be abandoned.
f. An induced step can be abandoned only by abandoning its inducing step.

7. Applying The General Graph Model to PSDL

The PSDL (Prototyping System Description Language) prototyping method uses a hierarchical decomposition strategy for filling in more details at any level of a prototype design. It uses stepwise, topdown refinement to selectively refine and decompose critical components. Each higher level component is described in terms of lower level components and the relations among them. The decomposition of each level is a realization of the components at a lower level of detail [53]. This hierarchical structure of the PSDL prototyping method is compatible with the underlying data model defined for the evolution history graph and introduced in section III.B above.

PSDL has been designed to prevent implicit interactions between modules, thus supporting module independence. The state of a state-machine operator is purely local in PSDL. One cannot send a data stream directly to a composite operator's component because the component names are not visible outside the implementation part of the composite operator.

This locality makes it easier to modify PSDL prototypes because the number of modules affected by a change are limited and can be determined by a straightforward mechanical analysis [54]. This is because when a composite module is decomposed into sub-modules, the implementation of the composite module uses its own specification and the specifications of the sub-modules, but not the implementation of the sub-modules [52]. This limits the impact of evolution steps and indicates that each "part_of" relation between a module and its sub-modules implies a "used_by" relation between the specification of the sub-modules and the implementation of the module. This also implies that the "used_by" relation can serve as the basis for automatic identification of inputs of a proposed evolution step and identification of induced steps triggered by a proposed step.

The "used_by" relation is not only defined between PSDL components but also between the requirement hierarchy and the corresponding PSDL components that fullfil each requirement. This means each PSDL component can be traced to a requirement in the requirement hierarchy and vice versa. This implies that a change in a system requirement can automatically generate a proposed evolution step to modify the specification of the corresponding component which, if approved by management, can generate the corresponding induced steps of this change to the specification/implementation modules that use the original specification.

A change request is normally represented by an evolution step that can be applied to one primary input component. However, as a natural result of the explicit relations among the modules in PSDL, it becomes obvious that the implementation of the change in one component can lead to changes in several system components. This leads to changing the atomic step into a composite step and producing a new atomic step for each of the affected components. These new atomic steps are called induced steps while the step inducing them is called the inducing step. As a result, the inducing step becomes a composite step and the induced steps become its substeps.

In order to eliminate the possibility of unnecessary relations between composite steps and their substeps, the composite evolution step may not produce any new component by itself, i.e., it is an empty step.

Figure 8 shows what we mean by unnecessary relations between a composite step and its substeps. In Figure 8.a step S1 is the composite step with the primary input A, steps S2 and S3 are two induced steps with primary inputs B and C respectively that uses module A. For step B or C to start step A has to be completed which is impossible because S1 is composite step and it will not be completed unless S2 and S3 are completed (circular dependency). In Figure 8.b, S1 is designated as an empty step with S2, S3, and S4 as its substeps that have the same relations as those between S1, S2 and S3 in Figure 8.a. This way S2 can be committed permitting S3 and S4 to start, when S3 and S4 are completed, S1 is committed producing a new configuration of the whole software system incorporating this change.

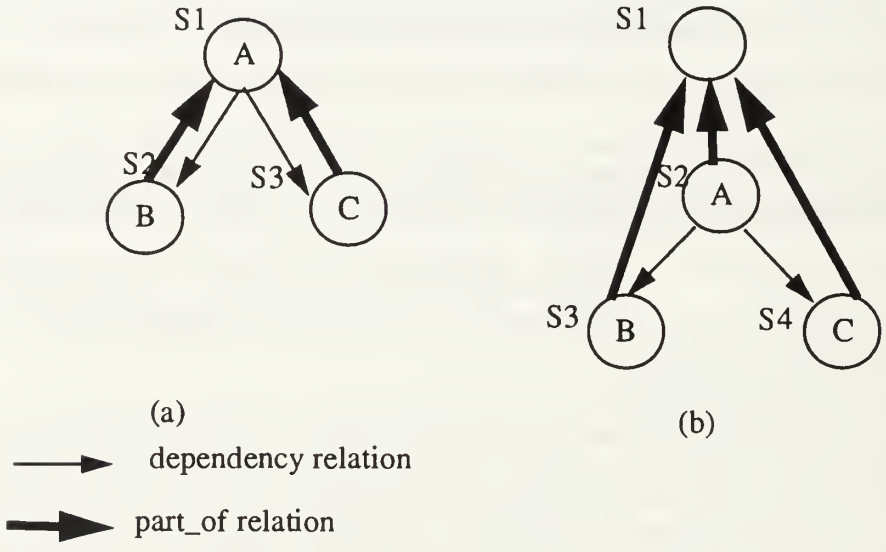


FIGURE 8. Relation between composite step and its substeps

The composite step decomposition occurs when the step is approved by the manager. This triggers the automated step analysis process that analyzes the relations between the primary input of each step and the rest of the system components, determines the initial approximation of the affected modules (the set of modules that use the output of the step, which can be approximated by those modules that use the primary input of the step) that have to be modified to propagate the required changes, and creates a sub-step for each of the affected modules. These induced steps behave no differently than any other step

and they may have relations with one another or some other evolution steps. Since the composite step itself is an empty step, there are no dependency relations between the composite step and its sub-steps.

C. SCHEDULING MODEL

The task in our case is to schedule a set of N evolution steps $S = \{S_1, S_2,..., S_N\}$ relative to a set of M designers $D = \{D_1, D_2,..., D_M\}$. The designers are of three possible expertise levels {Low, Medium, High}. Each step has associated with it a processing time tp (S_i), a deadline d (S_i), a priority p (S_i), and required expertise level e (S_i). Steps have precedence constraints given in the form of a directed acyclic graph G = (S, E) such that $(S_i, S_j) \in E$ implies that S_j cannot start until S_i has completed.

Because of the dynamics of the prototyping/evolution process, the steps to be scheduled are only partially known. Time required, the set of sub-tasks for each step, and the input/output constraints between steps are all uncertain, and are all subject to change as evolution steps are carried out.

Our goal is to dynamically determine whether a schedule (the time periods) for executing a set of evolution steps exists such that the timing, precedence, and resource constraints are satisfied, and to calculate this schedule if it exists.

1. Scheduling Constraints

During software system evolution/prototyping, constraints that reflect the importance and partial ordering of implementing evolution steps arise from real life situations. These constraints influence the evolution process and they must be represented in the scheduling model in order to reach a realistic schedule. The set of constraints that affect scheduling evolution steps in the context of our model are the following:

1. Precedence constraints.
2. Timing constraints.
3. Priority constraints.
4. Resource constraints.

The **precedence** constraint is used to reflect the inter-dependencies between the inputs and the outputs of evolution steps. The intent of this constraint is to impose a sequential ordering between given pairs of steps. The precedence constraints among evolution steps are represented in the form of a directed acyclic graph G = (S, E) such that $(S_i, S_j) \in E$ implies that S_j cannot start until S_i has completed.

The dependency relation between evolution steps implies a precedence constraint between these steps. These relations can be formalized using the dependency graph definition above and considering C as the set of system components as follows:

$$ALL(c : C, s_i, s_j: S \mid [s_i, c] \in O \ \& \ [c, s_j] \in I :: (s_i, s_j) \in E) \qquad (14)$$

These dependency relations mean that an atomic evolution step cannot start unless the module that needs to be changed is available as well as all the modules that affect this module. Most of the precedence relations are calculated automatically. Additional ordering constraints can be added manually by the manager, due to considerations such as a designer with a special skill is due to be assigned to a different project or that one step will be easier or less uncertain if some other step is carried out first.

The precedence attribute of a step, defined as the set of steps that precede this step, is used to resolve conflicts when two steps S_1 and S_2 are bound to the same generic primary input. It is also used to determine which version of a secondary input component to a step S_1 is used when this component is specified by generic reference and its current version is a primary input to another step S_2. If S_2 precedes S_1 then the secondary input of S_1 will be bound to the output of S_2 and $(S_2, S_1) \in E$, otherwise it is bound to the current version (the primary input of S_2).

The **timing** constraints of a step are specified in terms of two parameters: the deadlines, the time by which the step must be completed according to customer restrictions or manager's resource planning, and the estimated duration, a management estimate of the time needed to perform the step.

The **priority**, a small positive integer, is assigned to a step to reflect the criticality of its deadline. The deadline of a higher priority step can be relaxed only if it cannot be met when the deadlines of all lower priority steps are removed.

The priorities of different steps should be compatible with the precedence constraints between these steps, i.e. no lower priority step can precede a higher priority step:

$$\text{If } (S_2, S_1) \in E \text{ implies that } p(S_2) >= p(S_1) \quad (15)$$

$$\text{If } (S_2, S_1) \in E \ \& \ p(S_1) >= p(S_3) \text{ implies that } p(S_2) >= p(S_3) \quad (16)$$

The ECS system should enforce these constraints and warn the manager to make the necessary changes (change either the precedence or the priority of the step) to comply with these constraints.

The only **resource** constraint of a step in our system is specified in terms of either of two parameters: the expertise level required to perform the step as one of three levels (low, medium, high) defined for the members of the design team, or by specifying a certain team member to perform the step.

In addition to automatically generated precedence constraints, the precedence, timing, priority, and resource constraints are assigned manually by the manager and may be changed during the evolution process according to the state of the system's evolution and external constraints as well as to resolve schedule conflicts.

2. Dynamics - What Can Change

Since one of the main goals of this system is to support incremental replanning as additional information becomes available, it is important to define what kind of additional information can be added/updated, the impact of such changes on both the scheduling and assignment processes, and how this may also impact the goal of minimizing wasted design efforts due to these changes. The candidates for change during the evolution/prototyping process are the following:

1. The primary inputs.
2. The secondary inputs.
2. The affected modules.

3. The precedence, priority, and deadline.
4. Task decomposition
5. The estimated_duration for performing each step.
6. The design team members.

In the following subsections each of these change candidates is discussed.

a. Primary Input Changes

Normally, an evolution step has one primary input except in case of merging different versions of the same component. In this case an adjustment (addition or deletion) of a primary input may be needed. This change has no additional effect on the schedule because it does not affect any of the other components, since all the step attributes (mainly secondary_inputs and affected_modules) should have been calculated using the original primary input.

b. Secondary Input Changes

Due to the ongoing concurrent changes and adding/deleting of components to/from the system, dependencies that did not exist in the previous version may be introduced, and dependencies that did exist may be removed. These dependencies are reflected in each step by a set of secondary inputs (all the components that are used by its primary input) which has to be changed to reflect any dependency changes by adding new secondary inputs to, or deleting existing secondary inputs from the secondary_input_set of the step. The additional impact of this change depends on the status of the step.

1. If the step status is "proposed/approved" then the change has no additional effect.
2. If the step status is "scheduled" then the impact in this case includes the following:
    a. modifying the dependency graph to reflect these changes.
    b. recalculating the schedule according to the modified graph.
3. If the step status is "assigned" then the impact in this case includes all the changes mentioned in 2 above as well as the following:
    a. suspending any assigned steps that become dependent on any uncommitted steps.
    b. assigning any new steps that become ready.
    c. sending immutable copies of the new secondary inputs to the corresponding designer.

Note that before the ECS suspends a step it warns the manager that the current change leads to the suspension of the step. This gives the manager a choice of either confirming the change or binding the modified secondary input to an older version so that no suspension occurs.

c. Affected Modules Changes

This is used to add/delete affected modules to a step after reviewing the automated analysis due to missing relations in the configuration graph that the system does not consider during this analysis. The impact of this change depends on the status of the edited step.

1. If the step status is "approved" then the impact is adding/abandoning the corresponding step.
2. If the step status is "scheduled" then the impact in this case includes the effects mentioned in 1 above with the addition of:
    a. modifying the dependency graph to reflect these changes
    b. recalculating the schedule according to the modified graph.
3. If the step status is "assigned", then the impact in this case includes all the effects mentioned in 2 above as well as the following:
    a. suspending any assigned steps that become dependent on any newly added steps
    b. assigning any steps that become ready.

d. Precedence, Priority, and Deadline Changes

Precedence, priority, and deadlines are used by both the scheduling and assignment mechanisms to resolve conflicts regarding the dependency between different steps and establish a partial or total ordering among the approved steps. The impact of changes to any of these parameters depends on the status of the step.

1. If the step status is "proposed/approved" then the change has no additional effect.
2. If the step status is "scheduled" then the impact in this case includes the following:
    a. modifying the dependency graph to reflect these changes.
    b. recalculating the schedule according to the modified graph
3. If the step status is "assigned" then the impact in this case includes all the changes mentioned in 2 above as well as the following:
    a. suspending any assigned steps that become dependent on any uncommitted steps

b. assigning any steps that become ready.

Note that when changing the precedence or the priority of a step the compatibility between these two constraints should be checked according to rules (16), and (17).

e. Step Decomposition

The term step decomposition is used to explain the situations in which a designer is assigned an atomic step, but while carrying out his step decides to decompose the assigned component or change the existing component's composition. It is for these cases that the restriction on atomic steps to produce at most one component has been relaxed to permit atomic steps to produce zero or more output components.

After this modification to the graph model a designer can decompose his assigned component and work on its subcomponents within the estimated duration of his original component, or he has to change its estimated duration to avoid the warning of missing the estimated finish time. The ECS checks for such situations when executing the commit_step command issued by the designer. The following cases are considered when the designer commits his step:

1. The ECS looks first for the modified version of the primary_input component of the step, and commits it to the shared data space (configuration graph) creating a new version of the primary_input component of the step. Let us designate this output component as the main component and the primary_input component as the original component.
2. If the step output is not atomic (i.e., more than one modified component in the designer's private_workspace) then for each of these components the ECS does the following:
   a. If the component is part_of the main component, the designer is asked if he wants to commit this component and whether the work in this component is complete.
   b. If the component is to be committed, the work is complete, and the component is a part_of the original component then it is added to the configuration graph as part_of the main component and as the successor of the version that belongs to the original component.
   c. If the component is to be committed, the work is complete, and the component is not a part_of the original component then it is added to the configuration graph as part_of the main component and as the first version of itself.

d. If the component is to be committed and the work is not complete, then in addition to adding the component to the configuration graph as part_of the main component as mentioned in b and c above, a step is automatically created with this component as its primary input. This is the case with teamwork where a designer normally decomposes a component into its subcomponents, creating stubs for each created subcomponent then commits these subcomponents, automatically creating a new step for each of them.

e. For those components that are part_of the original component and not part_of the main component, the ECS asks if they should be deleted and delete them from the main component composition if confirmed by the designer.

As an example of step decomposition, Figure 9.a shows the typical case when a designer is assigned an atomic step S11 with component B as its primary input (this step is part_of a top level step S1). He did his modifications to B and did not touch its decomposition. The resulting system is A* with a new version B* of B. In Figure 9.b the designer changed the decomposition of B. He deleted F, added G, modified D, and kept E untouched. When the designer commits his work the system checks for the original component B, commits it, creating the new version B*, then commits D creating the new version D*, E is kept in the new system configuration A*, F is deleted, and G is added. If the work in G is not completed (according to the designer's response) a new step s12 is automatically created with component G as its primary_input. This new step (S12) is a sibling of step S11. When S12 commits its output will replace component G as part of component B*. When S1 is committed the new system configuration A* is automatically generated.

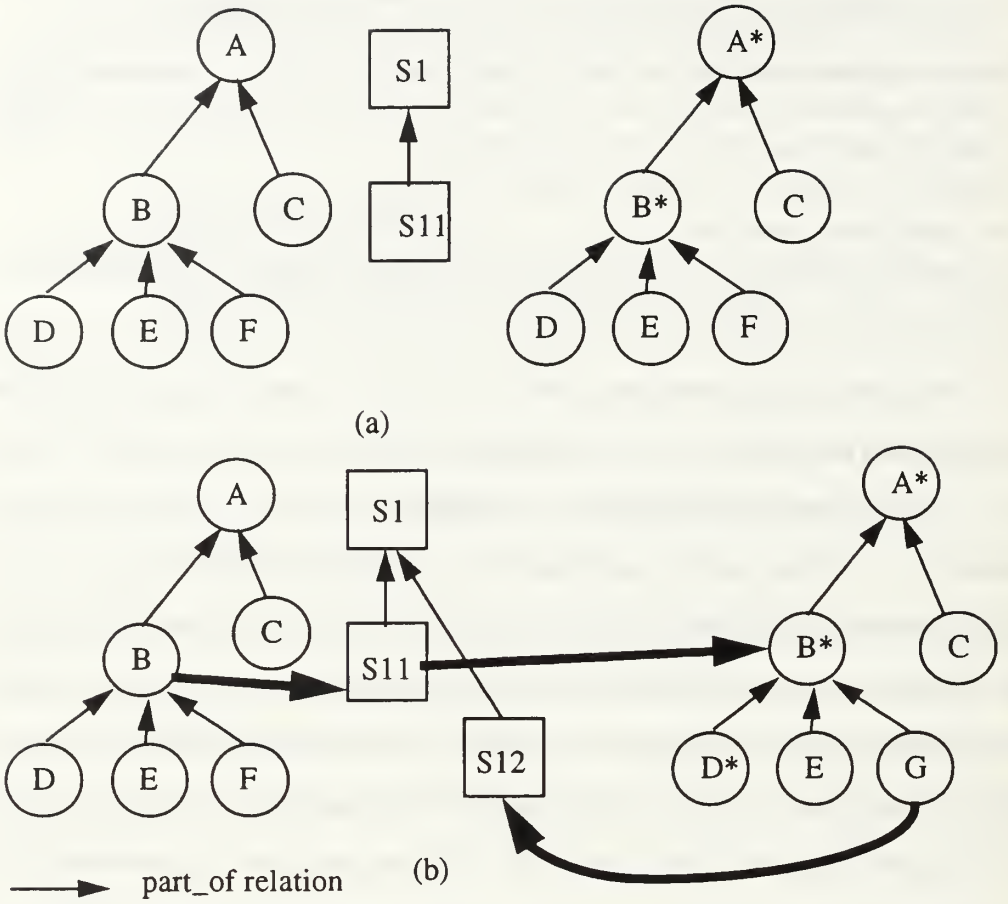


FIGURE 9. step decomposition

f. Time Estimate Changes

The time estimate is a management estimation of the time required for execution of each step. The initial schedule estimation is based on these time estimates. These estimates may differ from the actual time needed by the designers in charge of executing the steps. Editing these estimates may be needed due to actual time required for completed work or analysis of values of similar work, or for entry of time estimates for automatically generated sub-steps. This editing will provide a better scheduling estimate. The impact of these changes also depends on the status of the step.

1. If the step status is "proposed or approved" then the change has no additional effect.
2. If the step status is "scheduled or assigned" then the impact in this case includes recalculating the schedule.

g. Designer-Pool Changes

Three kinds of changes may take place in the designer_pool: add designer, delete designer, and change a designer's expertise level.

a. Adding a designer triggers the assignment mechanism to find an assignment for the new designer if there is one and the rest of the schedule is adjusted accordingly.
b. Deleting a designer has no impact if he is not assigned to a step and has no planned assignment in the schedule, otherwise the manager is prompted for confirmation, then if confirmed the assigned step is suspended and reassigned to some other designer and the rest of his planned assignment are rescheduled to the rest of the design team.
c. Updating the designer's expertise level has no immediate impact on the current assignment, but the planned assignments are adjusted according to his new expertise level.

h. Impact of Changes

When the impact of a change includes suspending an assigned step, leading to a rollback of some work done, or rescheduling some steps in a way that leads to missing any of the deadlines, the system should alert the manager to such consequences, giving him the option to continue with this change or cancel it and get back to the original conditions before the change. In the first case where a step has to be suspended, the system should prompt the manager with the candidate steps for suspension, how long they have been assigned, and the estimated working time. In the second case where some deadlines have to be missed, the system should prompt the manager with the candidate steps for missing their deadlines and the earliest completion time for each step compared with the completion time before the change. In both cases the manager should have the option to continue with the change or cancel it and continue with the original situation.

3. Computational Feasibility

Scheduling tasks with arbitrary precedence constraints and unit computation time in multiprocessor systems is NP-hard for both the preemptive and nonpreemptive cases [67] [84]. Scheduling nonpreemptive tasks with arbitrary ready times is NP-hard in both multiprocessor and uniprocessor systems [67] [83] which excludes the possibility of the existence of polynomial time algorithm for solving the problem. Hong and Leung [34]

proved that there is no optimal on-line scheduler can exist for task systems that have two or more distinct deadlines when scheduled on m identical processors where m > 1.

Scheduling evolution steps to more than one designer with arbitrary precedence constraints and arbitrary deadlines is a combination of both the multiprocessor scheduling problems mentioned above which is shown by many researchers to be NP-hard. These negative results dictate the need for heuristic approaches to solve scheduling problems in such systems.

D. RESOURCE MODEL

The only resource required in our model is the members of the design team. They are represented as a set $D= \{D_1, D_2,..., D_M\}$, where M is the number of designers in the design team. Each team member has associated with him an expertise level e (D_i) to reflect his expertise. The expertise levels represented are (low, medium, high). The resource constraint of a step determines what level of expertise is (at least) required of a designer who can be assigned to the step. Note: The manager may even require a specific designer to perform a certain step.

E. ECS MODEL

The purpose of the Evolution Control System, ECS, is to provide automated support for changes in plan during the execution of the plan, and provide automatic decision support for planning and team coordination based on design dependencies captured in the configuration model. The ECS also manages the software evolution steps from its creation to completion and provides automatic version control and configuration management for the products of these steps. Additionally, the system manages the design team through automation of the evolution steps' assignments to members of the design team in such a way that maximizes the team's effort, supports cooperative teamwork, and meets the management constraints such as precedence, priorities, and deadlines.

1. Context Model

The Evolution Control System (ECS) interacts with two external entities: the software evolution manager and the software designer. These represent classes of human

users rather than external software or hardware systems. There is one external interface for each class of user: the manager_interface and the designer_interface. Both of these interfaces are views of the proposed ECS. The message flow diagram in [10] and the stimulus-response diagrams in Figures 10, 11, 12 and 13 show the context of the system and the available commands, their effects and the possible error conditions.

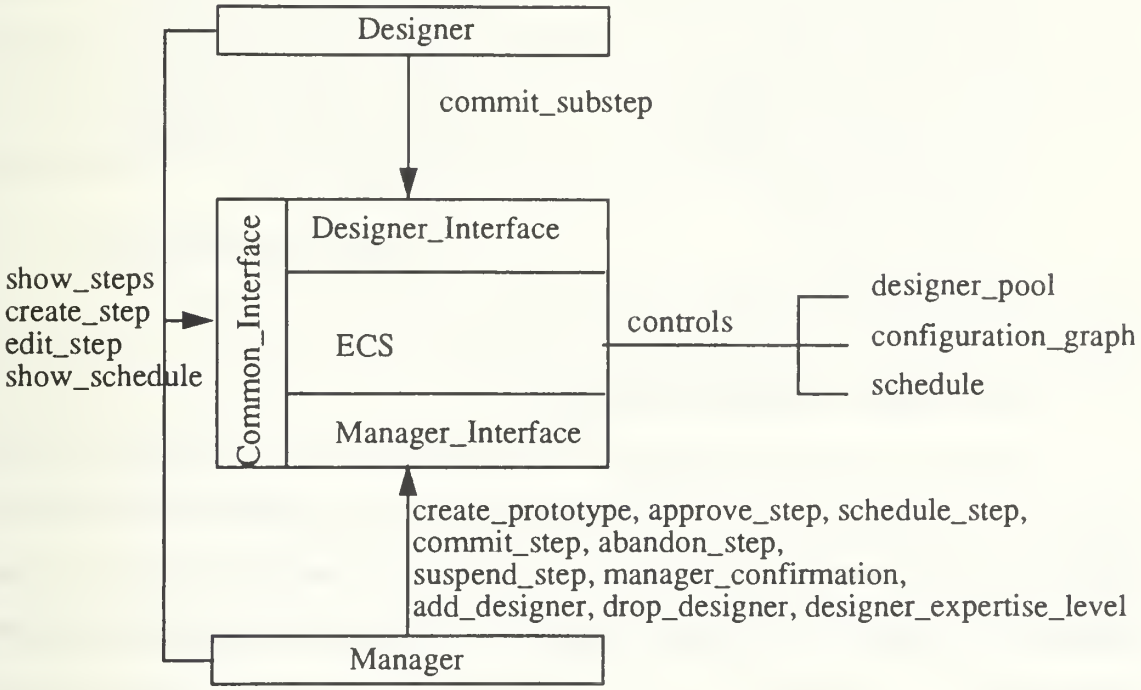


FIGURE 10. ECS message flow diagram

2. Event List

The event list for the ECS consists of 25 events. Most of the events are command-driven. The following is a list of the events; events triggered by the manager are marked M, events triggered by a designer are marked D, and common events for both designers and manager are marked C following the event name.

1. create_prototype (M)
2. create_step (C)
3. approve_step (M)
4. schedule_step (M)
5. commit_step (M)
6. commit_substep (D)

7. abandon_step (M) 8. suspend_step (M)

9. show_steps (C) 10. show_schedule (C)

11. add_designer (M) 12. drop_designer (M)

13. designer_expertise_level (M) 14. edit_step (M)

15. manager_confirmation (M)

The edit_step event is a set of events by itself as listed below:

1. add_primary_input 2. add_secondary_input

3. add_affected_modules 4. delete_primary_input

5. delete_secondary_input 6. delete_affected_modules

7. update_precedence 8. update_priority

9. update_deadline 10. update_estimated_duration

11. step_expertise_level

3. State Model And Related Concepts

The state of the ECS consists of a configuration graph, a schedule, a set of designers, and mappings giving the following attributes for each evolution step: deadline, estimated duration, precedence, priority, status and required expertise level. The formal definitions of the state model and the constraints on a feasible schedule are defined in Appendix A.1.

a. Configuration graph

As presented in section B, the evolution history is modeled as a graph G=[C, S, CE, SE, I, O]. This graph is a directed acyclic graph (bipartite with respect to the edges I and O). C and S are the two kinds of nodes (C: software component nodes, and S: evolution step nodes respectively). Each node has a unique identifier. C and S nodes alternate in each path that has only I and O edges. This represents the evolution history view of the graph. The edges represent the "part_of" (between a sub-component of a composite component and the composite component) and "used_by" relations (defined between components to represent the situation where the semantics or implementation of one component A depends on another component B; B used_by A) between the software

components of a given configuration ($CE \subseteq C \times C$), the "part_of" relation between a substep of a composite step and the composite step ($SE \subseteq S \times S$), the input relation between the system components which must be examined to produce output components that are consistent with the rest of the system and the corresponding evolution steps($I \subseteq C \times S$), and output relation between evolution steps and the components they produce ($O \subseteq S \times C$). System components are immutable versions of software source objects that cannot be reconstructed automatically. This graph is referred to from now on as the configuration graph. In Appendix A.1.a we formalize the structure of this graph as well as the two data types; components and steps that constitute its nodes

b. Designer_Pool

Designers are the only resource used by the ECS. The type designer is part of the ECS state model. Formal specifications of type designer is presented in Appendix A.1.b.

c. Schedule

The schedule in our system is a data structure representing the mapping between the steps and the scheduled designer to perform each step together with the estimated start and finish time. Formal specification of this data structure and the relevant concepts are presented in Appendix A.1.c.

4. Manager Interface

The manager interface to the ECS enables the manager to create new prototypes, provide for the evolution of the existing prototypes via a complete set of commands for creating, editing, scheduling, suspending/abandoning and/or committing evolution steps, and manage the designer_pool data via add_designer, drop_designer, and designer_expertise_level commands. The formal specifications of the various commands with the different responses for each command are defined in Appendix A.2 and Appendix A.3.

5. Designer Interface

The designer interface to the ECS enables the designer to view the steps in a given prototype with a given status and get the sub-steps assigned to him. This interface also enables the designer to create a sub-step of an assigned step as well as committing the assigned sub-step. The formal specifications of the various commands with the different responses for each command are defined in Appendix A.2 and Appendix A.4.

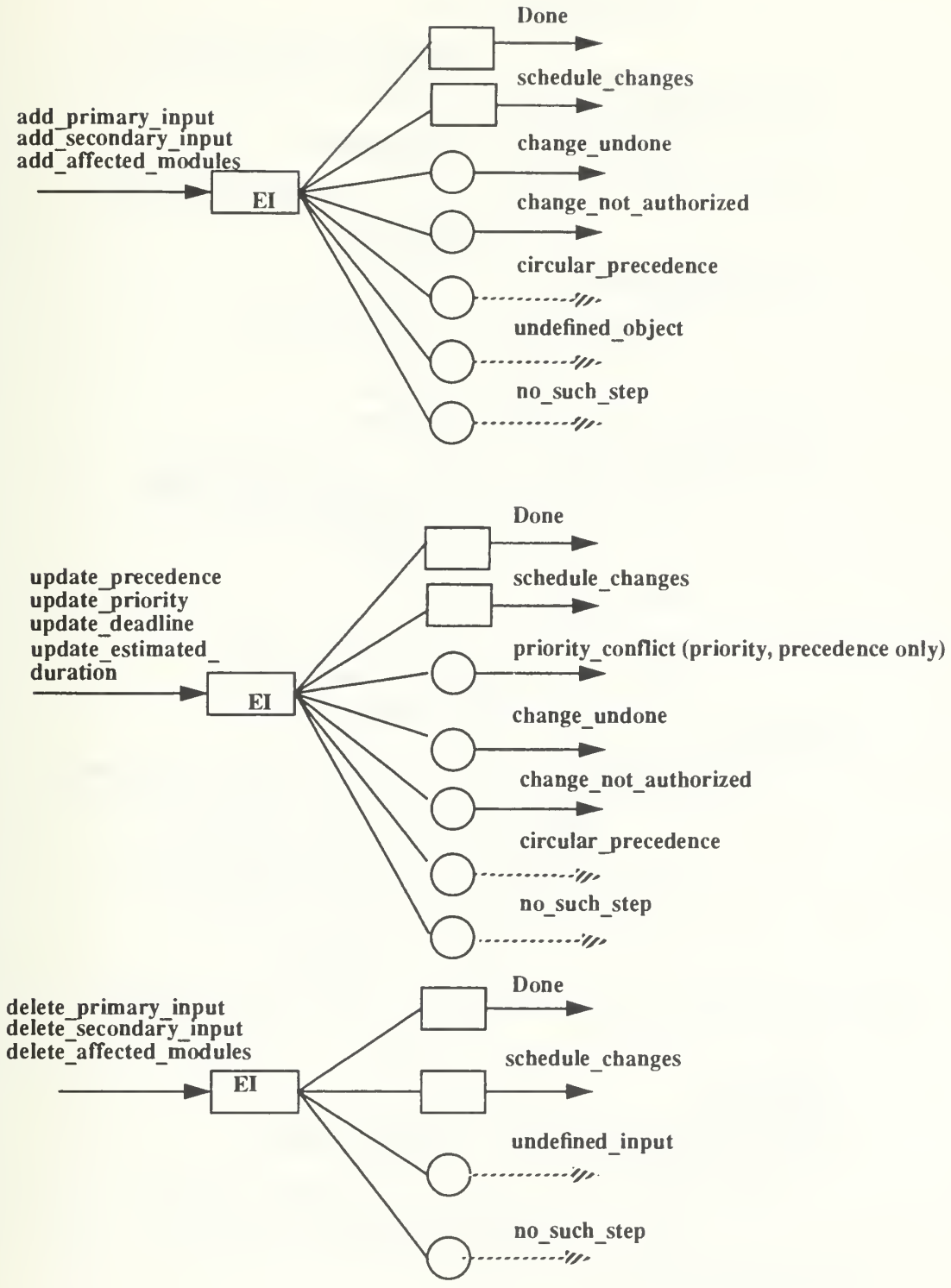


FIGURE 11. Stimulus Response diagram for the edit interface

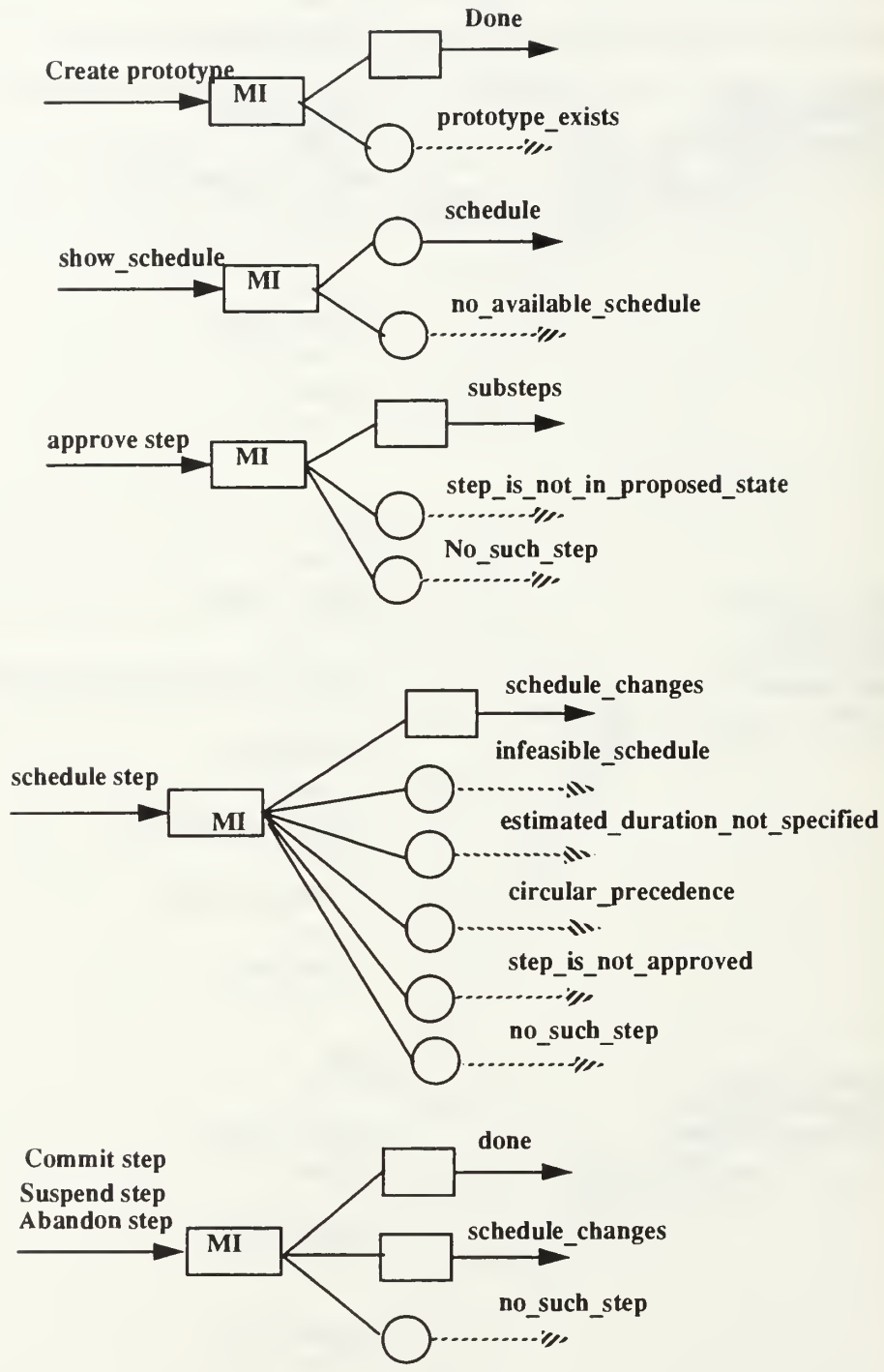


FIGURE 12. Stimulus Response diagram for the manager interface

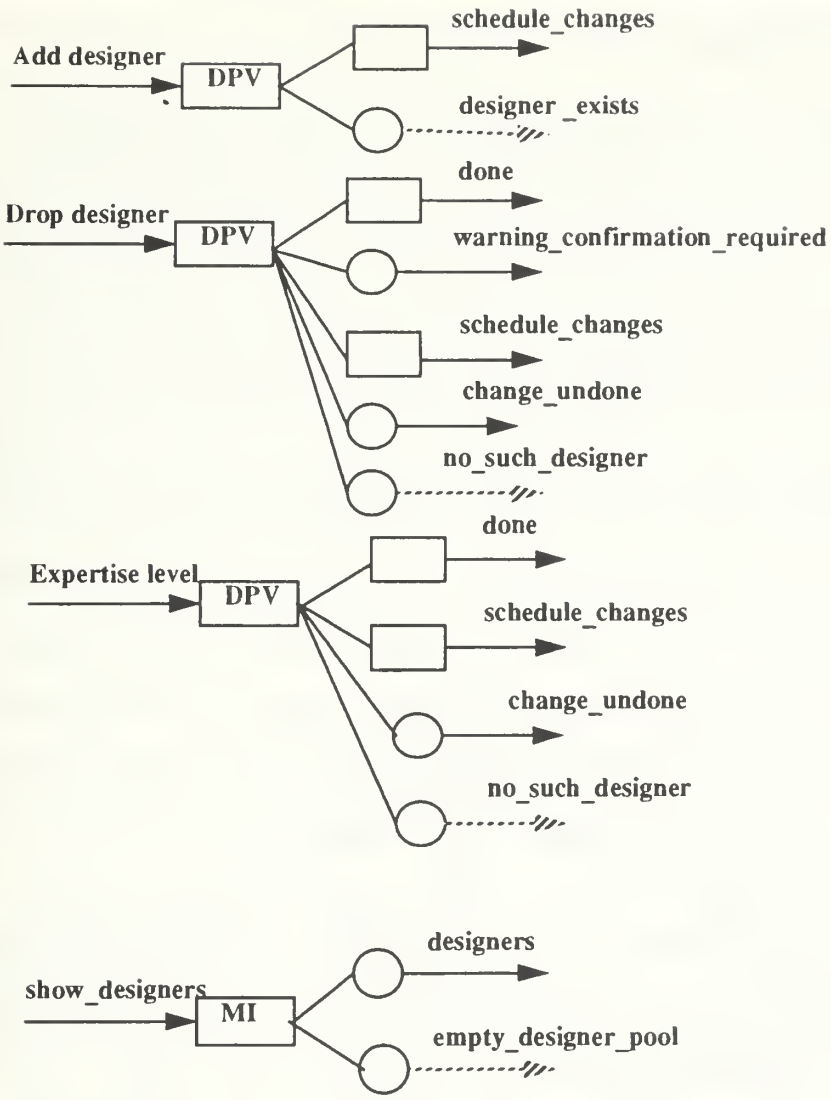


FIGURE 13. Stimulus Response diagram for the designer_pool view

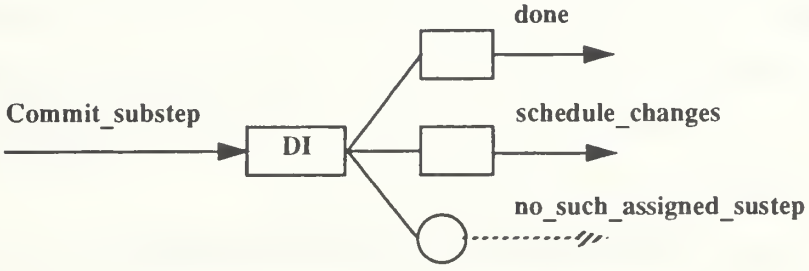


FIGURE 14. Stimulus-Response diagram for the designer interface

F. VALIDATION OF ECS SPECIFICATION

1. A Typical Scenario

Assume that we have a system Sys consisting of three main modules Ma, Mb and Mc as illustrated in Figure 15. Ma consists of two objects Oa, Ob and Mb is atomic while Mc consists of Oc, Od, and Oe. The relation "part_of" represents this hierarchical structure as follows: Ma part_of Sys, Mb part_of Sys, Mc part_of Sys, Oa part_of Ma, Ob part_of Ma, Oc part_of Mc, Od part_of Mc, Oe part_of Mc. The "part_of" relation also implies a "used_by" relationship between the implementation module of the parent component and the specification modules of the children components. The used_by relation for the example is: every specification module of a component is used_by its implementation module such as Sys.spec used_by Sys.imp, Ma.spec used_by Ma.imp, etc., and Oc.spec used_by Oe.imp. The used_by relation between Oc and Oe can only arise in PSDL if a timer is defined in Sys or Mc, this timer is started, stopped, or reset in Oc and the value of this timer is read by Oe. This kind of dependency should occur rarely in practice. In figure 3 a thin arrow from A to B means A used_by B, while a thick arrow from A to B means A part_of B & A used_by B.

Assume our design team has three members: Designer d1 with expertise_level high, d2 with expertise_level medium and d3 with expertise_level low. One typical scenario of a change to this system can be described as follows:

1. Three evolution steps are created (using the create_step command) by the designers, based on user feedback from a prototype demonstration, as shown in Table 1. The system will assign a unique number to each of the created steps. The status of each newly created step is "proposed". Each created step is added to the configuration graph as an isolated node (at this point, the primary inputs of the steps are specified by generic references, since we have not yet determined which version of the primary inputs will be acted on by each step) with no input or output edges.

Create_step command also triggers the automated step analysis process that analyzes the relations between the primary input of each step and the rest of the system components, determines the initial approximation of the affected modules (the set of

modules that use the output of the step, which can be approximated by those modules that use the primary input of the step) that have to be modified to propagate the required changes. The analysis process is done as follows:

a. The system examines the image of the primary input modules under the "used_by" relationship to find out what other modules have to be modified to reflect and propagate the required changes. By examining the primary input to step s1 the system should find out that the only modules that use Ma.spec are {Ma.imp, Sys.imp}. The same analysis is done for both steps two and three, resulting in the two induced sets {Sys.imp} and { } respectively.

The ECS also calculates the secondary inputs of the step as the modules used_by the primary_input of the step.

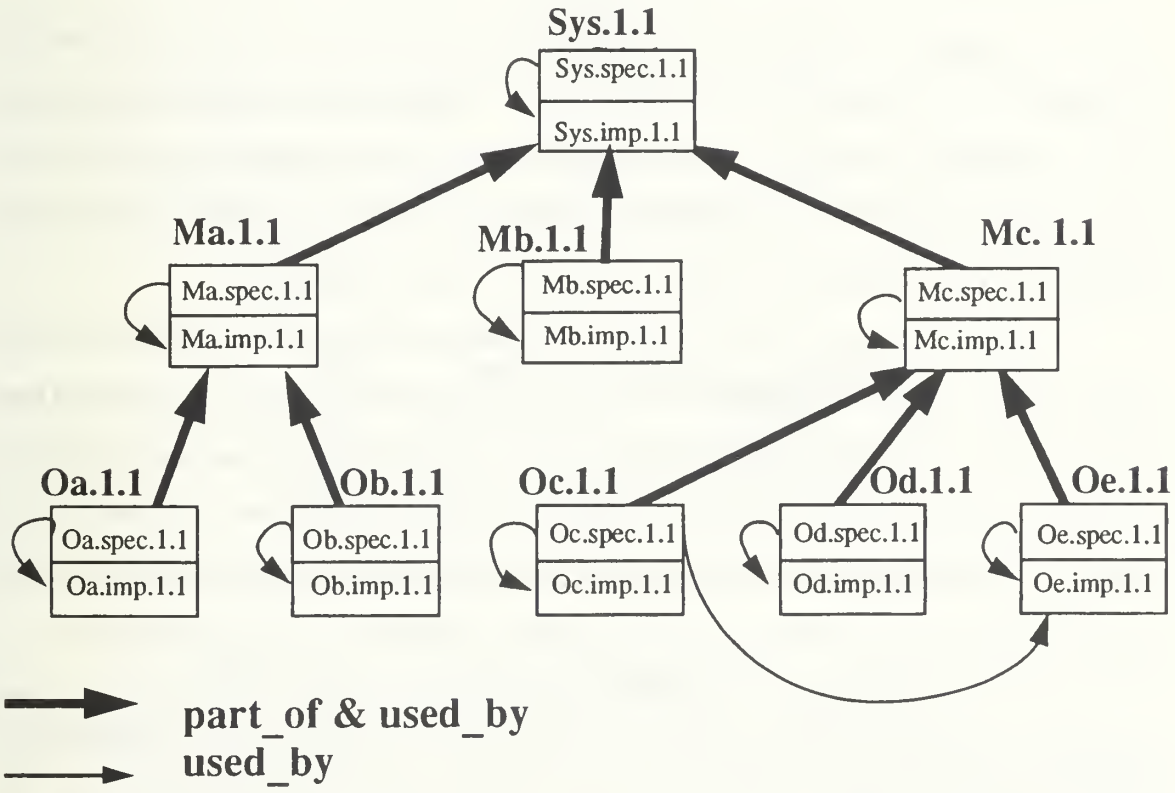


FIGURE 15. Version 1 of the given system.

2. The manager reviews the proposed steps and its calculated secondary inputs and affected modules where he can edit these automatically calculated attributes using the

"edit_step" command. The manager also, adds management constraints such as priority (type natural), precedence (set of preceding steps), estimated time for completing the step (hours), and deadline (time). Assume for the sake of this example that the manager added the values shown in Table 1 for these attributes. The manager also can modify any of the inputs to the step (using the edit_step interface). When the manager approves the step (using the approve_step command) its status is changed from proposed to "approved".

step_id	primary_ input	secondary _input	step_ expertise _level	precedence	priority	deadline	status
s1	Ma.spec.1	--	Medium		10	d + 16	proposed
s2	Sys.spec.1	--	High	s1	9	d + 18	proposed
s3	Oe.imp.1	Oc.spec.1 Oe.spec.1	Low		7	d + 20	proposed

TABLE 1. THE PROPOSED STEPS AND THEIR ATTRIBUTES.

3. Approving the step makes the system automatically create an atomic sub-step of the top level parent step for each affected module. Table 2 shows the different substeps created for each step. These sub-steps inherit the version bindings of their respective super-step as well as its precedence, priority, and deadline. These sub_steps are added to the configuration graph with part_of edges linking them to their super-steps.

4. The manager enters the estimated duration for each of the substeps. In this scenario, the estimated durations entered are the values shown in Table 2. The manager can then schedule the steps (using the schedule_step command), which triggers the scheduling and job assignment mechanisms. The schedule (including the planned assignment of a designer for each step and the estimated start and finish time for each step) is built incrementally with the issuing of the schedule_step command for each step as follows:

a. When step s1 is scheduled (using the schedule_step command), the manager gets a schedule which includes the substeps of step s1 as shown in Table 3. This schedule meets its deadline. The time T is the time when the schedule is produced. As indicated in Table 3, s1.1 is automatically assigned to d2, the status of s1.1 is changed to "assigned", the primary input of s1.1 is bound to Ma.spec.1.1, a mutable copy of Ma.spec.1.1 is sent to the private workspace of designer d2, and a message (e.g., an e_mail)

is sent to d2 telling him about his assignment. Notice that s1.2 and s1.3 are not assigned yet because they are dependent on s1.1. The assignment of s1.2 and s1.3 will be triggered by the commitment of s1.1 that makes both of them ready to be assigned.

step_id	substeps	primary_ input	secondary _input	step_exp ertise_le vel	estimated _duration	status
s1	s1.1	Ma.spec.1		Medium	7	approved
	s1.2	Sys.imp.1	Sys.spec.1 Ma.spec.1 Mb.spec.1 Mc.spec.1	Medium	6	approved
	s1.3	Ma.imp.1	Ma.spec.1 Oa.spec.1 Ob.spec.1	Medium	3	approved
s2	s2.1	Sys.spec.1		high	6	approved
	s2.2	Sys.imp.1	Sys.spec.1 Ma.spec.1 Mb.spec.1 Mc.spec.1	high	5	approved
s3	s3.1	Oe.imp.1	Oe.spec.1 Oc.spec.1	low	5	approved

TABLE 2. THE SUBSTEPS CREATED FOR EACH STEP

step #	designer	start_time	finish_time
s1.1	d2	T	T+7
s1.2	d2	T+7	T+13
s1.3	d1	T+7	T+10

TABLE 3. THE CURRENT SCHEDULE AFTER SCHEDULING s1

b. When step s2 is scheduled (using the schedule_step command) shortly after scheduling s1 (we can neglect the time difference between scheduling s1 and s2 relative to the time units used for estimating efforts due to the difference in magnitude), an updated schedule is calculated accordingly as shown in Table 4. Since the new schedule is feasible, it becomes the current schedule and it is sent to the manager. Having s2.1 ready to be assigned triggers the assignment mechanism that assigns s2.1 to d1. The status of s2.1 is

automatically changed to "assigned", the primary input of s2.1 is bound to Sys.spec.1.1, a mutable copy of Sys.spec.1.1 is sent to the private workspace of designer d1, and a message is sent to d1 telling him about his assignment.

step #	designer	start_time	finish_time
s1.1	d2	T	T+7
s2.1	d1	T	T+6
s1.2	d2	T+7	T+13
s1.3	d1	T+7	T+10
s2.2	d1	T+13	T+18

TABLE 4. THE CURRENT SCHEDULE AFTER SCHEDULING s2

c. When s3 is scheduled (using the schedule_step command) at t = T+3, an updated schedule is calculated accordingly as shown in Table 5. Since the new schedule is feasible, it becomes the current schedule and it is sent to the manager. The updated schedule is also feasible. Having s3.1 ready to be assigned triggers the assignment mechanism that assigns s3.1 to d3. The status of s3.1 is automatically changed to "assigned", the primary input of s3.1 is bound to Oe.imp.1.1, a mutable copy of Oe.imp.1.1 and immutable copies of the secondary inputs to s3.1 (Oe.spec.1.1, Oc.spec.1.1) are sent to the private workspace of designer d3, and a message is sent to d3 telling him about his assignment.

step #	designer	start_time	finish_time
s1.1	d2	T	T+7
s2.1	d1	T	T+6
s3.1	d3	T+3	T+8
s1.2	d2	T+7	T+13
s1.3	d1	T+7	T+10
s2.2	d1	T+13	T+18

TABLE 5. THE CURRENT SCHEDULE AFTER SCHEDULING s3

5. Since the three designers are informed about their assignment and the components required for each step exist in the corresponding designer's private workspace, the designers can start their assignments immediately. The rest of the assignments are triggered by either the commitment of an assigned step, which makes the immediate successor steps ready when there are idle designers waiting for tasks, or the availability of

a designer who has completed a previous task when there are other tasks ready to be assigned. Assuming a situation where everything goes as scheduled, the following actions take place:

a. When d1 issues the commit_step command at t =T+6, s2.1 is committed producing Sys.spec.1.2 and adding the corresponding node and edge to the configuration graph, the private workspace of designer d1 is cleared and he becomes idle. Committing s2.1 and having d1 idle triggers the assignment mechanism that finds no ready step for him.

b. When d2 issues the commit_step command at t =T+7, s1.1 is committed producing Ma.spec.1.2 and adding the corresponding node and edge to the configuration graph, the private workspace of designer d2 is cleared and he becomes idle. Committing s1.1 makes s1.2 and s1.3 ready to be assigned, it also triggers the assignment mechanism that assigns s1.2 and s1.3 to d2 and d1 respectively, automatically copying both the primary and secondary inputs of each step to the corresponding designer's private workspace, changing their status to "assigned", and sending both of them messages about their new assignment.

c. When d3 finishes his assignment at t =T+8 and issues the commit_step command, s3.1 is committed which means adding the output of s3.1 (Oe.imp.1.2) to the configuration graph (shared data space) as a component node, and the corresponding edge (s3.1, Oe.imp.1.2) as an output edge. The private work space of d3 is cleared and d3 becomes idle. Committing s3.1 as the only substep of s3 triggers the automatic commitment of s3 producing the new system version Sys.1.2 as illustrated in Figure 16. Having designer d3 idle triggers the assignment mechanism that finds no ready step to be assigned to d3.

d. When d1 issues the commit_step command at t =T+10, s1.3 is committed producing Ma.imp.1.2 and the corresponding node and edge are added to the configuration graph, the private workspace of designer d1 is cleared and he becomes idle. Having d1 idle triggers the assignment mechanism that finds no ready step for any of the idle designers.

e. When d2 issues the commit_substep command at t =T+13, s1.2 is committed producing Sys.imp.1.2 and the corresponding node and edge are added to the configuration graph, the private workspace of designer d2 is cleared and he becomes idle.

Committing s1.2 makes s2.2 ready to be assigned which triggers the assignment mechanism that assigns s2.2 to d1. Committing s1.2 as the last substep of step s1 triggers the automatic commitment of s1 producing the new system version Sys.1.3 as illustrated in Figure 17. Similar actions take place when committing s2.2 and that lead to the automatic commitment of step s2 producing the new system version Sys.1.4 as illustrated in Figure 17.

This typical scenario illustrates that the ECS provides the necessary commands needed to create, approve, schedule and commit an evolution step, as well as add its management constraints, and trigger the assignment mechanism. The managerial functions of determining the affected modules of a change and automatically creating evolution steps for propagating these changes are automated. The manager retains the option to override any automatic calculations through the edit commands (add and delete affected_modules) and may direct the continuity of actions by his approval via the approve_step command. The planning process is automated via the scheduling mechanism which is triggered by the schedule_step command. This scheduling mechanism finds a feasible schedule if one exists or gives suggestions to the manager in order to get a feasible one, while giving the manager the option to accept ECS suggestions or modify the management constraints to accurately reflect all the management concerns.

The automatic assignment of the ready steps to the idle designers as well as the triggering actions of these assignments are also illustrated. The ECS simplifies the designer's task by sending copies of all the components he/she needs to complete his task (primary and secondary inputs) in addition to a message telling him about the new assignment that resides in his private workspace. As soon as the designer finishes his task he can directly issue the commit_step command without having to worry about versioning his component or controlling the configuration it belongs to since the version and configuration control functions are automated and transparent to the designer.

This scenario also illustrates the incremental planning nature of the system where new steps can be added to the schedule when they are available, and automatically scheduled either by finding a feasible schedule if one exists or providing suggestions and

accepts the manager's inputs to fine tune the management constraints to reach a feasible one.

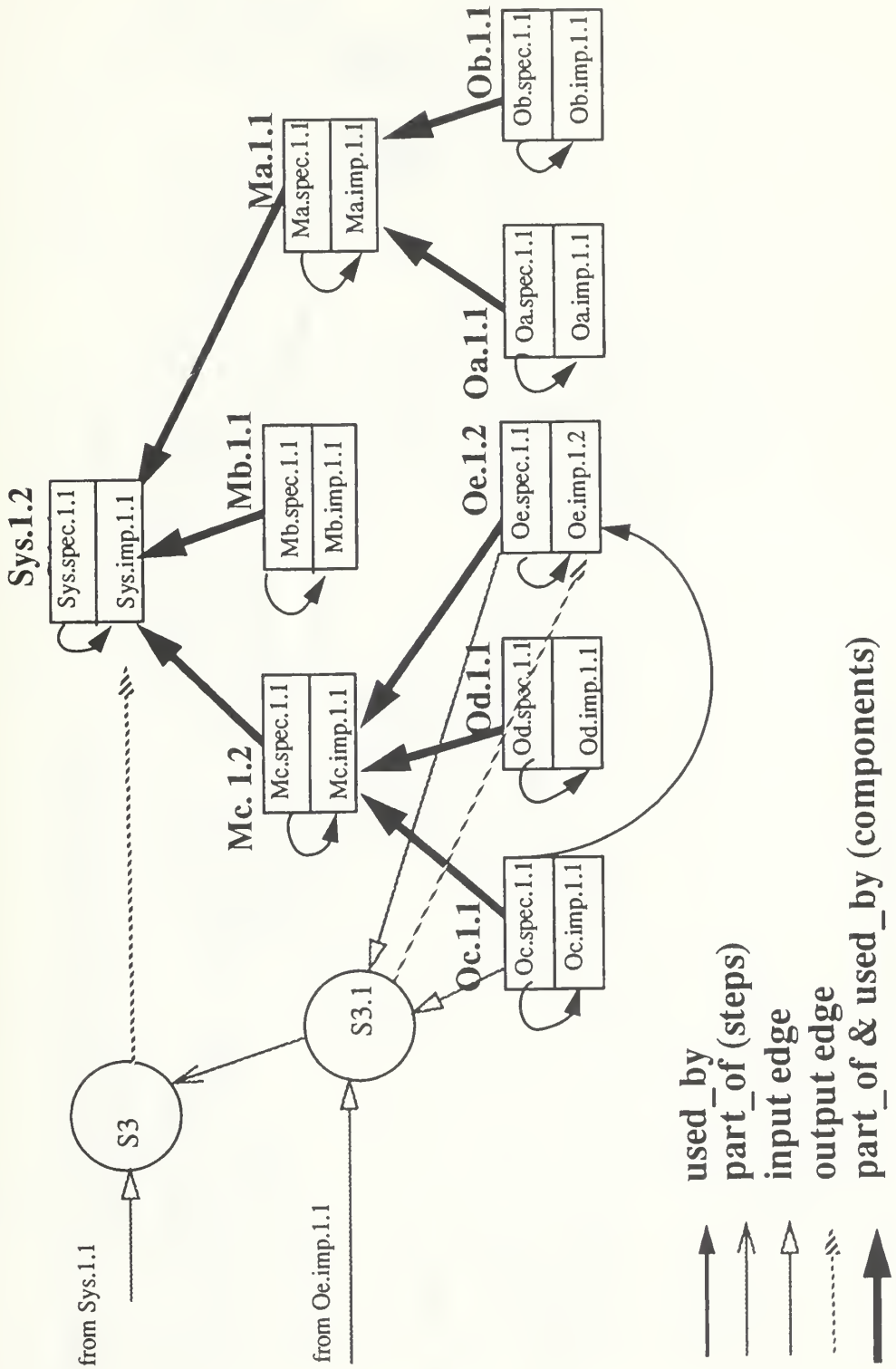


FIGURE 16. Version 2 of the given system.

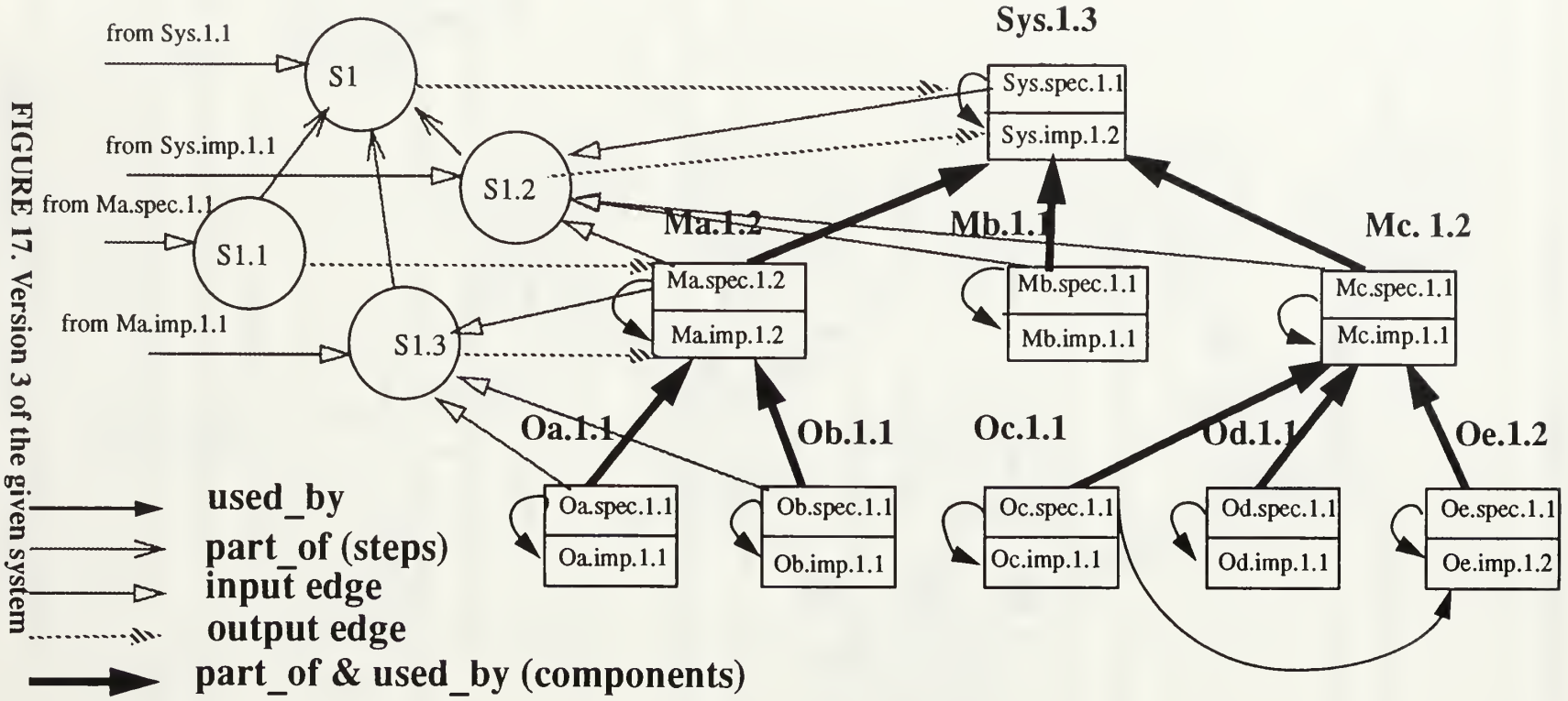


FIGURE 17. Version 3 of the given system

FIGURE 18. Version 4 of the given system

2. Scenario for a Missed Estimated Finish Time

Now consider that our example does not go so smoothly and designer d2 was unable to finish his assignment s1.2 by its estimated finish time t = T+13. The assignment mechanism should alert the designer that the estimated finish time has passed and ask for formulation of a new estimate. The designer either responds by committing the step or entering a new estimated_duration for this step. The schedule is then recalculated. If the new estimated_duration is 9 instead of the original value 6, then the schedule is recalculated as shown in Table 6.

step #	designer	start_time	finish_time
s1.1	d2	T	T+7
s2.1	d1	T	T+6
s3.1	d3	T+3	T+8
s1.2	d2	T+7	T+16
s1.3	d1	T+7	T+10
s2.2	d1	T+16	T+21

TABLE 6. THE CURRENT SCHEDULE AFTER THE NEW ESTIMATED_DURATION OF s1.2

In this schedule s2 has missed its deadline. Note that s2.2 depends on s1.2. and s1.2's new finish time is t = T+16 and s2.2 needs 5 time units to finish. the manager gets a warning that s2 will miss its deadline and the suggestion of a new deadline to be equal to the estimated finish time.

The system in this case monitors the estimated finish times and warns the corresponding designer if the finish time of his task is reached without issue of the commit_step command. The system also warns the manager in this case only if the designer chooses to change the estimated duration of his task and this change leads to missing any of the deadlines of the steps and provides the manager with appropriate suggestions.

3. Scenario for an Early Commit

Now consider designer d2 finishes his assignment s1.2 after 4 time units instead of 6 time units. In this case the system will not wait until t = T+ 16 to assign s2.2, but s2.2

is assigned right away giving it a chance for early completion, and the schedule is adjusted as shown in Table 7.

In this case the positive effect of finishing a task early may lead to a ripple of positive effects for the whole schedule if the next task for the designer who finished early is ready. If the next assignment is not ready or no next assignment exists for the designer, the schedule will not be affected. As an example, if d3 finishes s3.1 at t = T + 6, the schedule will not be affected because there is no other ready step that requires a low expertise_level.

step #	designer	start_time	finish_time
s1.1	d2	T	T+7
s2.1	d1	T	T+6
s3.1	d3	T+3	T+8
s1.2	d2	T+7	T+11
s1.3	d1	T+7	T+10
s2.2	d1	T+11	T+16

TABLE 7. THE CURRENT SCHEDULE AFTER THE EARLY COMMIT OF s1.2

4. Scenario for Changing the Precedence of a Step

Using Table 5 as our final schedule, a possible change that can occur as the schedule is carried out is changing the precedence of one or more steps. Consider that the manager decided to change the precedence of step s1 from none to be preceded by step s2 and the precedence of s2 to be preceded by none (using the update_precedence command). As soon as a precedence changes the system checks two conditions for conflict: 1) that the new precedence does not lead to a circular dependency between the scheduled steps and 2) that there is compatibility between the priority of this step and the priorities of the steps that precede this step and warn the manager accordingly.

In our case changing the precedence of s2 produces no warning messages when it happens. Changing the precedence of s1 leads to different warnings according to when it takes place. If the precedence of s1 is changed before that of s2, the manager gets the warning "circular_precedence" (at this point s1 precedes s2 and s2 precedes s1) which will be eliminated by changing the precedence of s2. When this warning is fixed or the precedence of s1 is changed second the manager gets the warning "priority_conflict"

because s2 now precedes s1 and priority of s1 is greater than that of s2. This conflict is solved by changing the priority of s1 to the value 8 (using the update_priority command). This last change is also checked for conflicts and none are found.

The impact of this change on the current schedule depends on when such a change takes place. Let us consider two cases where the impact is different as an example of the capabilities of the ECS. In the first case we consider the situation where the change takes place before t = T+ 7 and the second one is at t = T+8.

a. Changing Precedence Leading to an Infeasible Schedule

When the precedence change occurs before commitment of s1.1 at t = T+7, the manager gets the warning "infeasible_schedule" (because s1 misses its deadline) and the suggestion that the deadline of s1 should be changed to 17 instead of 16. Since this value is the lower bound (it is the total execution time of three dependent steps), the manager has the option of either to accept the system's suggestion or to undo the change and keep the original feasible schedule. The resulting schedule when the manager accepts the system's suggestion is shown in Table 8.

step #	designer	start_time	finish_time
s2.1	d1	T	T+6
s1.1	d2	T	T+7
s3.1	d3	T+3	T+8
s2.2	d1	T+6	T+11
s1.3	d2	T+7	T+10
s1.2	d2	T+11	T+17

TABLE 8. SCHEDULE AFTER CHANGING THE PRECEDENCE OF S1, S2 BEFORE T+7

b. Changing Precedence Leading to Infeasible Schedule and Step Suspension

When the precedence change occurs at t = T+8, after committing s1.1 and assigning s1.2, step s1.2 becomes dependent on s2.2 and s2.2 is not yet assigned. The ECS sends a warning to the manager that s1.2 has to be suspended. At this point the manager has the option either to reject the suspension of s1.2 and, as a result, the precedence change is undone and the original schedule is restored, or confirm the suspension of s1.2. In the latter

case, the ECS calculates the updated schedule which includes suspending s1.2 until s2.2 is assigned and completed. As a result, s1.2 is automatically suspended by sending a message to its designer telling him that the step is suspended, clearing his private workspace, and sending him his new assignment if there is one. The manager then gets another warning that the updated schedule is infeasible (s1 misses its deadline). Again the manager has the option to either accept the updated infeasible schedule or to tune the other management constraints to get a feasible one.

step #	designer	start_time	finish_time
s1.1	d2	T	T+7
s2.1	d1	T	T+6
s3.1	d3	T+3	T+8
s1.3	d1	T+7	T+10
s2.2	d1	T+10	T+15
s1.2	d2	T+15	T+21

TABLE 9. SCHEDULE AFTER CHANGING THE PRECEDENCE, CASE 2

This scenario illustrates how the ECS keeps the manager informed of the consequences of his decisions and gives him the necessary data to take appropriate action. It also shows how a step is automatically suspended after manager confirmation without costing him or the designer any additional effort. This facilitates streamlining of both of their tasks.

5. Scenario for Changing the Decomposition of an Assigned Task

Let us consider again the typical case where everything goes smoothly as scheduled in Table 5. This time when s1.1 is assigned to d2 he finds out that the module Ma.1.1 has to be re-decomposed. The new decomposition keeps Oa after modifying it, adds a new component Of, and deletes Ob. Now d2's private workspace includes the modified components Oa.spec.1.1, Of.spec and Of.imp (new components do not have version or variation numbers yet). When d2 commits his step, using the commit_step command, The following actions take place:

a. The modified component Ma.spec.1.1 (the primary_input) is configured (calculates its version and variation number) as Ma.spec.1.2, and added to the configuration graph (shared data space).

FIGURE 19. Version 3 of the system after task decomposition

b. The modified components Oa.spec.1.1, Of.spec, Of.imp are committed as Oa.spec.1.2, Of.spec.1.1, and Of.imp.1.1. Two proposed steps are created with primary inputs Oa.spec.1.2, Of.spec.1.1 respectively.

c. Committing s1.1 requires d2's interaction to change the secondary input set to s1.3 to include Of.spec.1.1 and delete Ob.spec.1.1.

d. When the top level step s1 commits later, the part_of relation of the new version Ma.1.2 should be modified to include Oa.1.2, Of.1.1, and exclude Ob.1.1 as shown in Figure 16.

6. Assessment of the Adequacy of the Proposed Command Set

To assess the adequacy of the proposed command set for the purpose of the ECS, we walk through the different functions and operations required by the system and check whether the given command set is sufficient to perform each one of them. The best way to do that is to start from scratch where we have no prototype in the environment. The create_prototype command enables both the manager and the designer to create new prototypes with unique names as a logical first step. The evolution of this prototype is done via evolution steps. The creation of such steps is done using the create_step command which leads to a creation of a step in the proposed state. The management control over ongoing evolution activities is enforced via the approve_step command. This command controls the steps accepted for the system's consideration, and gives the manager the ability to enter and update the management constraints (precedence, priority, deadline, and estimated_duration) as well as the step attributes (primary, secondary, and affected modules) via the corresponding commands update_precedence, update_priority, update_deadline, update_estimated_duration, add/delete primary_input, secondary_input, and affected_modules. The schedule_step command is a manager command for incremental planning of approved steps when they become available either after fulfillment of its management constraints or having a budget to implement it. The continuous control of the manager over the steps in progress (scheduled or assigned) is guaranteed via the suspend_step, and abandon_step commands that enable him to suspend a step and get it back to the approved state or abandon the step completely as dictated by various management situations. The manager's confirmation is also a condition to any automated system decisions concerning suspending a step according to a new dependency inserted into the system, or slipping a deadline to get a feasible schedule. The manager in both

previous cases is given the choice to either confirm the system decision or use the set of edit commands at hand to adjust various attributes of the step or the management constraints to reach a decision that accurately reflects the management's concerns. The set of commands add_designer, drop_designer, and designer_expertise_level enables the manager to control the designer pool.

Designers tasks are streamlined with the ECS. The designer interface has two main commands. The create_step and commit_step commands enable him to create a step according to prototype demonstration feedback or a change request from the customer site, and to commit his completed work of his assigned step without having to worry about the versioning of his committed components since the version and configuration control are transparent to him. The designer does not need any other commands since the system sends to his private workspace all the data required to do his task including the primary and secondary inputs of the step.

7. Questions and Design Decisions

1. Is the designer responsible for a composite step also responsible for its sub-steps?

No, otherwise each top level evolution step must be done by a single designer. This is not acceptable because team effort must be supported for rapid updates.

2. Can the ECS assign more than one step to the same designer at the same time?

No. The main reason for not doing that is to minimize the development time via maximum concurrency. This goal could only be reached if we let each designer concentrate on one task and get it done in the minimum time to enable the system to assign more tasks. This is due to the fact that a step can only be assigned if it is atomic or if all the steps it depends on are committed. This guarantees the consistency of the software system via change propagation and minimized roll-backs.

3. Is keeping a step history (recording the sequence of versions of an edited step) of value to the design/prototyping process?

Yes, this may help in tracing changes to a step, determining what kind of change to a step are most likely to happen, the effect of each kind of change on the system releases and deadlines, and accordingly help decide what kinds of step editing should be permitted; e.g. avoid step editing that has adverse effects on step completion. The main uses of this information are to improve the tools and process guidelines, and to evaluate the abilities of managers and designers. This capabil-

ity would not be used directly in the operation of the ECS. For this reason, we do not consider it further in our requirements and design.

IV. DESIGN DEVELOPMENT OF ECS

A. MODELING THE DESIGN DATABASE

The data in the design database includes all the components of the ECS state as given by the functional specifications in Chapter III.F, and Appendix A.1. These components are: the configuration graph, the current schedule, and the designer's pool. The configuration graph, G=[C, S, CE, SE, I, O], consists of two sets of nodes, C and S, the software component nodes and evolution step nodes, and four sets of edges, CE, SE, I, and O, the different kinds of edges representing the relations between pairs of evolution graph nodes component-component, step-step, component-step and step-component respectively. The design database (DDB) schema as well as the mapping between the components of the ECS state model to this schema is presented in the rest of this section. The implementation details of this schema using Ontos database [63] is given in Appendix B. The Ada interface to the design database schema which contains the definition of an Ada package (both specification and body) that enable the main ECS program (in Ada) to access the shared data in the design database according to the DDB schema, is given in Appendix C.1.

1. Design Database Schema

The main types defined in our database schema, shown in Figure 20, are:

1. Object: represents the persistence property.
2. Version: a subtype of type Object to represent immutable versions of software objects.
3. Component: a subtype of type Version to represent the component decomposition of a prototype.
4. Top_component: a subtype of type Component to represent prototypes.
5. Step: a subtype of type Version to represent software evolution steps.
6. Top_step: a subtype of type Step to represent top level evolution steps.
7. Designer: a subtype of type Object to represent designer information.
8. Assignment: a subtype of type Object to represent a schedule record.
9. Text_Object: a subtype of type Object to represent software component text.
10. Sequencer: subtype of type Object to generate sequential numbers.

Top level evolution steps and their substeps are represented by instances of the types Top_step and Step respectively. The steps are represented in the database as a set of Step object with a sequencer object associated with this set to produce a unique step_id for each created step. Prototypes and their components are represented by instances of the types Top_component and Component respectively, and each defined versioned component has a corresponding Sequencer to keep track of the number of variations splitted for each component. The designer pool is represented by a set of instances of type Designer while the schedule is represented by a list of instances of type Assignment. The set of text files (.spec, .imp, .ps, .graph, and .a) belonging to each component are represented by instances of type Text_Object. The abstract representations of the different types are defined in the following subsections.

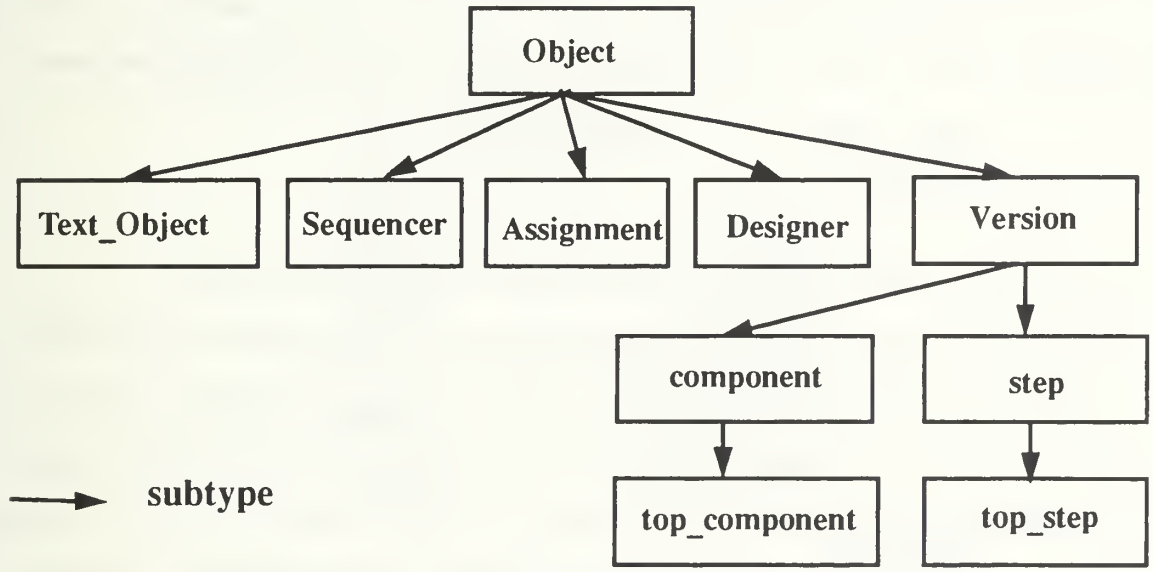


FIGURE 20. Type hierarchy

a. Type Object

The type object is the most general type. All other types are either directly or indirectly subtypes of type object. All instances of type object are persistent by definition. This means that entities that are created by an operation persist beyond the lifetime of this operation.

Type: object supertype: None

Properties: persistent

Name: string

Operations:

getObject

putObject

Both of these operations takes an object name as an input and returns a pointer to an object or saves an object in DDB respectively.

b. Type Version

All evolving objects are directly or indirectly subtypes of the type version. This type as well as its subtypes are immutable versions of software source objects that cannot be reconstructed automatically such as source code modules, specification modules, requirement modules, and evolution steps. Type version has the following abstraction:

Type: version Supertype: object

Properties:

version_id	: natural
variation_id	: natural
previous_version	: version
next_version	: version
time_created	: time

Operations:

get_previous_version

get_next_version

These two operations are used for history tracking as well as for the merge process to locate a common base version for a set of merging components. Each of these operations takes an object name as an input and returns a pointer to the previous or next version of this object.

c. Type Component

This type is the specification of type version for the immutable software components mentioned in type version above. Each instance of this type represents a frozen version of a software component.

Type: component Supertype: version

Properties:

created_by	: name
part_of	: set {component}
subcomponents	: set {component}
used_by	: set {component)
data	: text (represents component data)

-- list of text files (.spec, .imp, .ps, .graph, and .a files)

Operations:

create_component

retrieve_component

show_components

The operation create component takes a component name as an input and creates an object with this name in the design database (if this component does not exist in the current working directory of the DDB), it also adds the different text files from the designer workspace as the data of this object and it also assigns a version and variation number to the newly created object.The operation show_components is a directory like listing of the component and its subcomponents, while the operation retrieve_component includes copying the component to a specified workspace.

d. Type Top_component

top_component is a component that is not part of any other component. This type is used to represent prototype configurations and to enable distinguishing prototypes as a separate class to optimize their access time. This separation enables the iteration over the top_component (s) without having to iterate through all the components to locate only the top level ones.

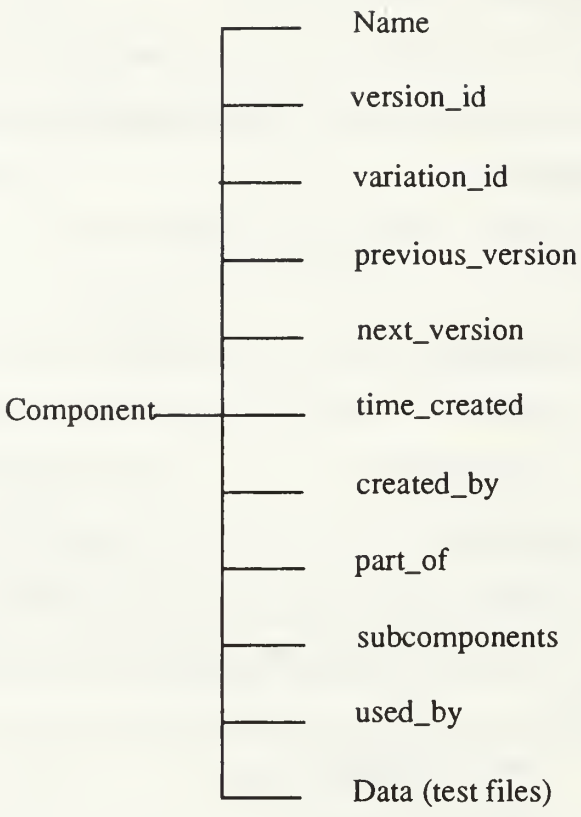


FIGURE 21. Component attributes

Type top_component Supertype: component

Properties:

part_of: { }

e. Type Step.

A steps is the evolution history node that has all the informations required to evolve a software component from one version to the next. Each instance of this type represents an evolution step or substep, composite or atomic. The type step is a persistent object.

Type: step Supertype: version

Properties:

primary_input : set {component}

secondary_input : set {component}

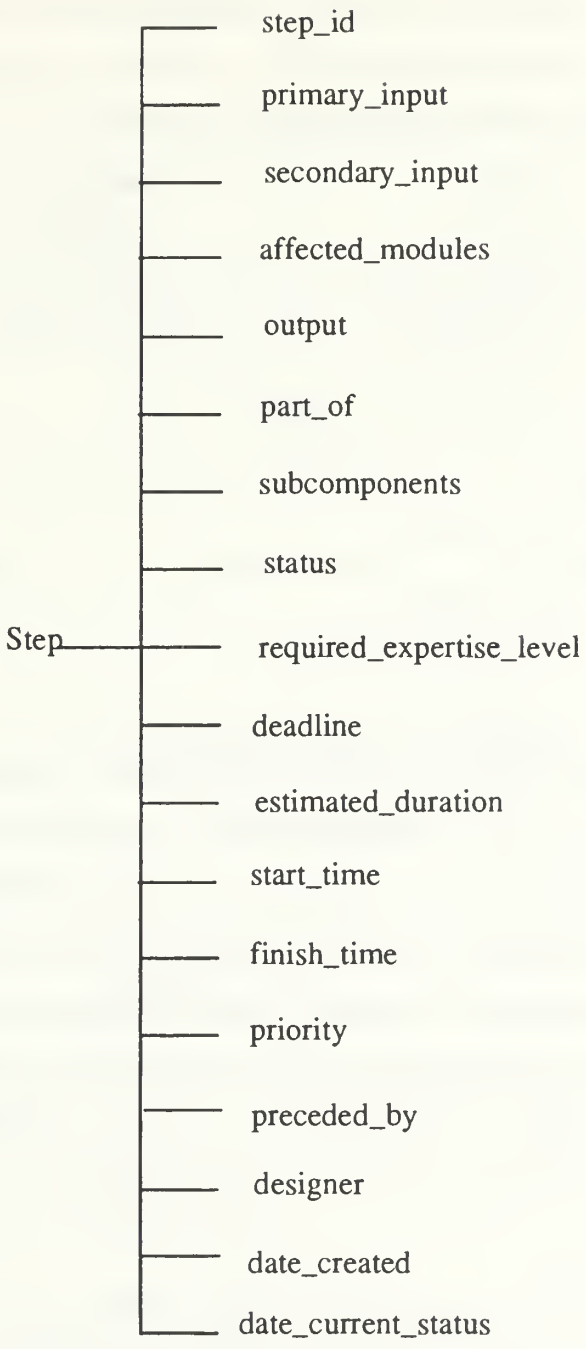


FIGURE 22. step attributes

output	: set{component}
part_of	: set {step}
status	: enumeration
required_expertise_level	: enumeration
deadline	: time

estimated_duration : natural (in hours)

estimated_start_time : time

estimated_finsh_time : time

priority : natural

preceded_by : set {step}

designer : name

date_created : date

date_of_current_status : date

Operations:

create_step

show_steps

show_step_details

update_step

The operation create_step takes a prototype name (with variation and version numbers) as a base version and a component name as primary input and creates step with a unique number. The operation update_step takes step number and arbitrary number of the step attributes, resets the single valued attributes to the given new values and add or delete a given value to the multivalued attributes. Show_steps operation takes as input an identifier of a certain category of the steps (such as top, proposed, approved .etc.) in the database and return a listing of those steps. Show_step_details operation takes a step number as an input and returns the different attributes of the step.

f. Type Top_step

This type is used to distinguish top level steps from their substeps to enable the iteration over the top steps without having to iterate through all the steps to locate only the top level ones.

Type: top_step Supertype: step

Properties:

part_of: { }

affected_modules : set{component}

subcomponents : set {step}

Operations:

show_substeps (listing of the substeps of the step)

g. Type Designer

This type is used to represent a designer's data in the design database. The designer's pool is represented by a set of instances of this type.

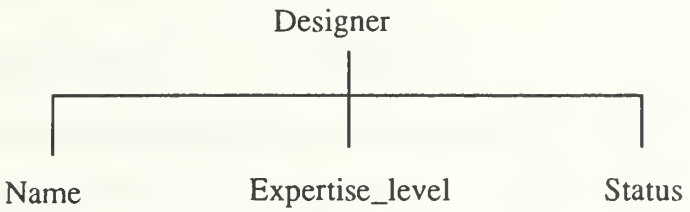


FIGURE 23. Designer attributes

Type: designer Supertype: object

Properties:

name: string

expertise_level: exp_level

status: enumeration {busy, free}

Operations:

add_designer

delete_designer

change_expertise_level

change_status

The designer's operations are adding new designer with a specified expertise level to the DDB, deleting a designer from the DDB, and changing the expertise level or the status of an existing design team member.

h. Type Assignment

This type is used to represent the relevant information of assigning a step to a designer. While the information included in this type may be redundant with respect to both designer data in the designer's pool and corresponding assigned step, this type optimize the access time to collect such information. This type also enables saving the planned assignment where a full schedule can be represented as a list of instances of this type.

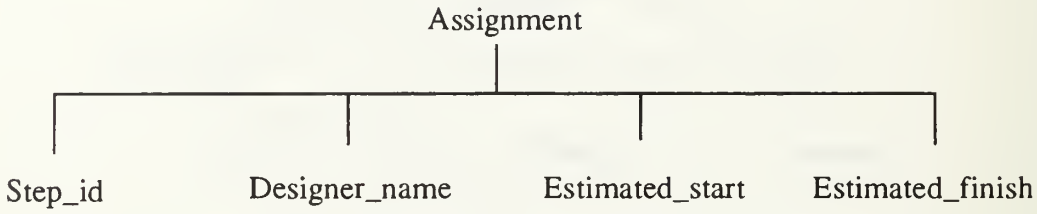


FIGURE 24. Assignment attributes

Type: Assignment Supertype: object

Properties:

step_id	: natural
designer_name	: string
estimated_start_time	: time
estimated_finsh_time	: time

Operations:

createAssignment

deleteAssignment

showAssignment

i. Type Schedule

An instance of this type represent the full schedule generated by the scheduler to be kept in the database.

Type: Schedule Supertype: object

Properties:

Assignments : List (Assignment)

Operations:

createSchedule

deleteScheule

showSchedule

The schedule uses a list as a container class for the assignments to enforce any ordering required over the schedule assignments.

j. Type Sequencer

Instances of this type are used as persistent counters where needed. An instance of this type is needed for each versioned object to keep track of the number of variations for each object. An instance of this type is also used to keep track of the number of steps created and to be used for steps unique numbering.

Type: Sequencer Supertype: object

Properties:

Value : natural

Operations:

getValue

IncrementValue

This is the only two operations needed on a sequencer and they can be implemented as one since any time the value is needed it has to be incremented.

k. Type Text_Object

Instances of this type are used to represent different files associated with each component in the design database.

Type: Text_Object Supertype: object

Properties:

Name :string

Value : Text

Operations:

createTextObject

R_retrieveTextObject

W_retrieveTextObject

getTextObjecName

The operation createTextObject is used to create a text object that has the name of the input text file and its value is equal to the text of the file. The operations R_retrieveTextObject and W_retrieveTextObject are used to get the text object from the database in read only mode or read/write mode respectively. The last operation getTextObjecName is used to list the names of the text objects included in a component

2. Concurrency Control

Concurrency control is one of the problems in the field of database management that has received much discussion and many solutions. Most of these solutions are based on two basic assumptions. First, all the transactions must be executed such that their results are equivalent to some serial execution of those transactions. Second, objects have a single value. In some of these solutions old versions are used for a short period of time as a transient states, but when the transaction completes the values of the old versions are not retained [85].

In the context of software evolution/development all old versions of an object are made available via the version control mechanism. This contradict the second assumption due to the nature of the design database, where a version of an object cannot be modified but can be read at any time (read-only). This makes it possible for readers and writers to work on the same object and never conflict. The immediate effect of this is an increase in concurrency level of the system.

The traditional question of what happens when two steps try to modify the same object occurs in our system in two cases: first, when the designer/management decides to explore/start another alternative which automatically splits a new variation of the original one with the alternate object version attached to it. Second, when two modifications have to be done to the same object, the serialization is automatically done by the Evolution Control System, in a higher level of abstraction, which takes care of planning these changes to be serialized according to a predefined management policies.These policies should not

permit starting one of the steps before committing the other one according to the management constraints such as precedence, and deadlines.

Figure 25 depicts the horizontal view of the graph representation of the relation between system versions and the corresponding evolution steps. Step S_k is applied to version $V_{i,j}$ of a software object (where "k" is the step number, "i" represents the variation number and "j" represents the version number along one variation) producing version $V_{i,j+1}$. Variations are represented as partial paths in the graph, applying step S_{k+1} to $V_{i,j+1}$ produces the version $V_{i,j+2}$ on the same variation line. Applying step S_{k+2} to $V_{i,j+1}$ produces a new variation i+1 with version $V_{i+1,j+2}$. Applying step S_{k+3} to $V_{i,j}$ produces another new variation i+2 with the version $V_{i+2,j+1}$.

The graph can also include dependencies between the modified versions and versions of other objects that are not modified by the step, such as specifications of other modules. For simplicity links of this type are not shown in Figure 25.

Notice that, in case of a split that creates a new variation, it is the order in which the step is assigned rather than the version number of the base version that decides the variation number (i. e., Step k+2 created the new variation i+1 and Step k+3 created the new variation i+2 despite the fact that the first is applied to version j+1 and the second is applied to version j). Thus the variation numbers capture the chronological order in which the variations were created. Step numbers are assigned in increasing order when the steps are created. Steps can be carried out concurrently and asynchronously, and the order in which they actually start or complete their implementation phases can differ from the order in which the steps are added to the schedule.

As for the other shared components of the ECS such as the designer-pool and the schedule, the modification to the former one is only performed by the design manager while the later is subject to change through the invoked instances of the designer_interface to ECS and the manager interface. For these components we rely on the underlying database concurrency control scheme (Ontos Database in our implementation) to serialize concurrent access to these components as an atomic transactions.

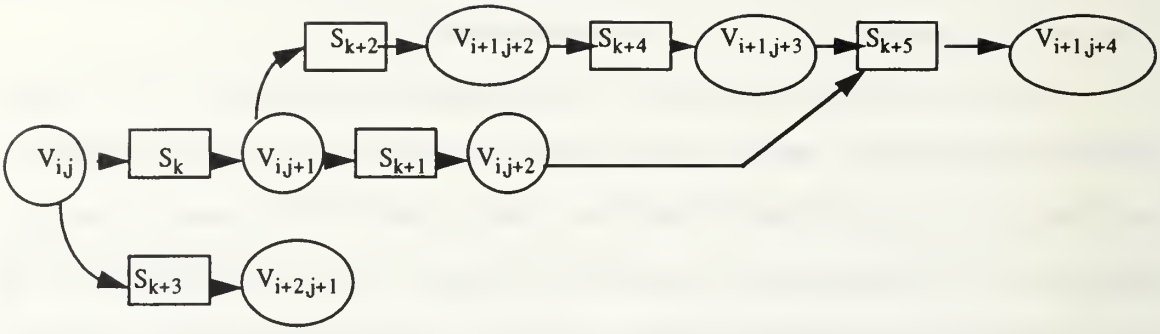


FIGURE 25. The relation between system versions and evolution steps

B. IMPLEMENTATION CONSIDERATIONS

The prototype implementation of the ECS is intended to be on unix system, where multiple instances of ECS are run concurrently by different users (managers and designers). These instances of the ECS share the ECS state data stored in the DDB as defined in section A above by the DDB schema and its Ontos representation defined in Appendix B.

1. Implementing Shared Data for Multiple Users

The shared data by multiple ECS users consists of the evolution graph (evolution steps and software component versions) the designer_pool data and the schedule. Despite the fact that the schedule information my be redundant with some of the steps attributes, it is stored as a separate component to optimize the access to the schedule informations a collected plan of a project implementation and to enable to relate different scheduled tasks to each other with respect to some scheduling attributes such as start time or expertise level and so on. The design database in the ECS implementation consists of two main parts: the first part is the shared data space where all the ECS state data are stored, and the second part is the designers work spaces where modification to software components are performed.

a. Shared Data Space

The shared data space is the repository that keeps all of the verified software objects (versions or configurations) and their corresponding evolution steps. The versions in the shared data space are frozen and may not be changed under any circumstances. Any changes to any of the objects must be done in the context of an evolution step, authorized

by the management, and completion of such a step can only add new versions to the shared data space. The shared data space contains the public releases of the software objects. Mutable copies of these objects can only be obtained as part of an evolution step controlled by the Evolution Control System. The relations between the objects in the database are kept as attributes of each object and the corresponding evolution step.

b. Private Workspaces

Since the data in the shared data space is frozen and may not be changed, the designers's private workspaces are used for production of new versions of existing objects or adding new objects to existing software systems which in turn produce new versions of the software system. A designer's private workspace is protected from updates by other designers while a step is in progress. This is visible via the other CAPS tools and has no impact on the observable behavior of the ECS system.

The private workspaces have the following relations with the graph model for software evolution:

1. There is a 1:1 correspondence between private workspaces and evolution steps. This means that each designer can only be assigned one step at a time.
2. There is a 1:1 correspondence between objects in the private workspace and the set of objects representing the inputs to and the outputs from the step of the workspace {inputs (s) U outputs (s): where s is the step of the workspace}.
3. Only the output objects of the step can have mutable copies in the private workspace.
4. Secondary inputs of the step have immutable copies in the private workspace

The process of copying objects to and from the designer's workspace is done automatically by the Evolution Control System (ECS), and these objects continue to be under its control until either all the changes are done and the ECS, triggered by the commit_step command from the designer, commits them to the shared data space producing a new version of each modified object (when a mutable version in the private workspace is committed by the ECS, it becomes an immutable version in the shared data space), or if the changes are suspended/abandoned then current copies of completed mutable versions are saved as an attribute to the step for future reference.

2. Choice of Languages and Support Systems

As a work done at the Naval Postgraduate School, computer science department, we do not have much of choice of the language to be used for implementing the ECS system, needless to say this language had to be Ada. However, the support object oriented database system is Ontos database [63] [64] that has its interface and schema written in C++. This adds the burden of having to define an Ada interface package that lets the ECS Ada program access the shared data in the DDB according to the defined DDB schema. Ontos DB is a multi-user, distributed object database with a C++ class library interface that provides a reliable storage facility for C++ objects. It has standard database capabilities and a special support for objects as well as a set of database and object oriented classes enhancing the power of the C++ language [64].

3. Software Decomposition and Structure

The Evolution Control System ECS consists of two main modules, the manager interface and the designer interface. The manager interface includes all the commands needed to implement the specified behavior of this interface given in chapter III.F and Appendix A. These commands include show_prototypes, show_steps, show_step_details, show_schedule, create_prototype, create_step, edit_step, edit_team, approve step, schedule_step, commit_step, suspend_step and abandon_step. The designer interface includes the commands needed by the designer to execute the assigned step which are create_substep, commit_substep and the adminstrative commands such as show_prototypes, show_steps, show_step_details, and show_schedule. The Ada implementation of these commands as defined in the functional specification is given in Appendix C.

C. THE SCHEDULING PROBLEM

Our problem is to schedule a set of sporadic tasks (software evolution steps). These sporadic tasks have random arrival times, and given deadlines, precedence constraints, and priority values to indicate the criticalness of their deadlines. Because of the unpredictable nature of the arrival time of the sporadic tasks, it is very difficult to design a real-time (on-

line) system that guarantees that all their deadlines can be met [34]. Moreover, each of these tasks requires certain expertise level, which implies that the system model is a set of M software designers of different expertise levels (not identical designers). This problem is similar to that of dynamic scheduling tasks with arbitrary arrival times, deadlines, and precedence constraints in a multiprocessor system where the processors are not identical.

Hong and Leung [34] proved that there is no optimal on-line scheduler can exist for task systems that have two or more distinct deadlines when scheduled on m identical processors, where $m > 1$. Scheduling tasks with arbitrary precedence constraints and unit computation time is NP-hard both the preemptive and the non-preemptive case [67]. Our problem is even more complicated than both of the above two cases, when contrasted with the case proven in [34] we have more than one designer and each step of the step set has its distinct deadline which is the same conditions for the conclusion reached by Hong and Leung, in addition, the designers are not of the same expertise level which makes it even more complicated. In contrast with results of [67] our problem includes arbitrary precedence constraints between pairs of the steps in the step set to be scheduled in addition to an arbitrary computation time for each step which makes it even harder than the case of having unit computation time. These negative results indicate the need for heuristic approaches solve this scheduling problem.

1. The Scheduling Algorithm

Scheduling a set of tasks to reach a feasible schedule is a search problem, where the search space can be structured as a search tree. The root of this search tree is an empty schedule, an intermediate node is a partial schedule, and a leaf node (terminal) is a complete schedule. Since not all leaves correspond to feasible schedules, it might cause an exhaustive search to find one, which is computationally intractable in the worst case. Because of the computational complexity of finding a full feasible schedule in many of the real applications, heuristic approaches are used.

a. System and Task Model

The task set in the ECS scheduling problem is a variable set of evolution steps $S = \{S_1, S_2,..., S_N\}$, where N varies with time. This set of tasks need to be scheduled to a set of M designers $D = \{D_1, D_2,..., D_M\}$. The designers are of L different expertise levels. Tasks (steps) are characterized by the following:

- Estimated processing time tp (S_i): a management estimate of the time required to perform a step.
- Deadline d (S_i): The time by which a step must be completed
- Earliest start time EST (S_i): the earliest time at which the step can be assigned to a designer (calculated when a scheduling decision is made).
- Priority p (S_i): An integer value to reflect the criticalness of the deadline of a step.
- Resource requirement r (S_i): required expertise level for performing a step.
- Precedence constraints given in the form of a directed acyclic graph G = (S, E) such that $(S_i, S_j) \in E$ implies that S_j cannot start until S_i has been completed.

In order to support teamwork, we assume that each step is assigned to a single designer. This designer must have at least the same expertise level as that of the step. We also define the earliest start time EST (Si) as the earliest time at which the step can be assigned to a designer. This time is calculated when a scheduling decision is made.

Our goal is to determine whether there exists a schedule for executing the tasks, that satisfies the timing, precedence, and resource constraints, and to calculate such a schedule if one exists. Since this problem is computationally intractable, we weaken the requirements to checking whether a feasible schedule can be found within the available time. Otherwise advise the software manager of the lowest priority deadlines that have to be canceled (moved to their calculated finish time) in order to get a feasible schedule. This algorithm should also give the software manager the choice to change other constraints such as priority, precedence or estimated execution time of the tasks to tune the schedule each time new evolution steps are to be added to the schedule and a feasible schedule cannot be reached. It also must check the validity of these changes (e.g. if a priority of a step is changed it has to be less than or equal the priorities of its predecessors and greater than or equal to that of its successors).

Thus, we need an on-line scheduler that is called when one or more sporadic tasks arrive at time t (new tasks in our system may have some of the constraints not defined when they arrive to the scheduler) or if the attributes of the currently scheduled tasks change, to decide if the newly arrived tasks, or the changed tasks, along with unassigned tasks at time t (scheduled but not started yet), could be rescheduled so that all deadlines are met. If a feasible schedule is reached the system will continue assigning the tasks to the designers according to the schedule constructed by the on-line scheduler. Otherwise the system will try to meet the deadlines of the most important (highest priority) tasks and suggest changing the deadlines of the least important ones. These suggestions could be accepted by the manager or he can change other parameters which in turn triggers the on-line scheduler to recalculate the schedule accordingly.

Changing the attributes of currently scheduled tasks means editing any of the constraints of the not-started-yet tasks, assigned tasks that are prone to exceed their estimated execution time (which is a common case in software effort estimation), and the addition/deletion of designers.

b. A Heuristic Search Scheduling Algorithm

A heuristic scheduling algorithm tries to reach a feasible schedule for a set of tasks by starting at the root of the search tree, which is an empty schedule, and tries to extend the schedule with one more task by moving to one of the nodes in the next level of the search tree until a feasible schedule is reached. The nodes in the next level of the search tree consist of those tasks that are ready to be scheduled, i.e. the tasks that have all their predecessors completed at this point or has no predecessors. A partial search path is extended only if it is strongly feasible.This is because if extending the current schedule by a task T causes T to miss its deadline then none of all the possible future extensions can meet the deadline of task T, since starting T later cannot make it finish earlier [67]. To this point we introduce the following definition:

- Strongly-feasible partial schedule: A partial schedule is strongly-feasible if all schedules reached by extending it by any of the remaining (ready to be scheduled) tasks are also feasible.

If the partial schedule is strongly feasible then a heuristic function is used to extend the partial schedule. This heuristic function should reflect various characteristics of the scheduling problem to effectively direct the search to a plausible path. If all the schedules resulting from extending the current schedule with any of the remaining tasks are also feasible, the partial schedule is called strongly feasible. The heuristic function is then applied to every task that is ready to be scheduled. The task with a predefined property of the heuristic function is selected to extend the current partial schedule (e.g. if we use the earliest deadline first as our heuristic then we pick the task with earliest deadline of the tasks that are ready to be scheduled to extend the current partial schedule), otherwise this search path is stopped because it will not lead to a feasible schedule.

Our heuristic algorithm is based on the heuristic algorithm introduced in [67] and discussed above. The main difference is that the tasks in our problem have precedence constraints which is not discussed in [67] where the authors deal with a set of independent tasks. Another difference is that each task has its own deadline rather than a common deadline for each set of tasks as is the case in [67].

Before describing the details of our algorithm, let us introduce the following definitions:

- Pending_step: a step whose predecessors (in the dependency graph) have all been scheduled (not necessarily assigned yet) and their estimated finish time is calculated. The step's earliest start time is set to the latest finish time of its predecessors.
- Ready_step: a pending step whose earliest start time is less than or equal the current time t.

The following data structures and variables are used by the algorithm:

- Dependency_graph: a directed acyclic graph G = (S, E) such that $S = \{S_1, S_2, ..., S_N\}$ is the set of steps to be scheduled, E is the set of edges such that $(S_i, S_j) \in E$ if and only if S_j cannot start until S_i has completed.
- In_degree: an integer representing the number of the immediate predecessors of each node (step) in the dependency graph.
- Pending_list: a list holding pending steps sorted in a non-decreasing order of their earliest start time.
- Ready_list: a list holding ready steps sorted in a non-decreasing order of the heuristic function used (e. g., deadlines, earliest start time etc.).

- Earliest Available Time (EAT): a vector of M values to represent the earliest available times of the resources (designers). EAT_i is the earliest time when D_i becomes available when the system has only one instance of each resource type (expertise level), e. g., for the case of having only three expertise level low, medium, and high and one designer of each level then EAT = (EAT_l EAT_m EAT_h). In case of having multiple instances of each expertise level the EAT is represented as a matrix so that each row represents the Earliest Available Times of the different instances of each expertise level.

$$EAT = ((EAT_{l1}\ EAT_{l2}\ ..\ EAT_{lk})$$

$$(EAT_{m1}\ EAT_{m2}\ ..\ EAT_{mr})$$

$$(EAT_{h1}\ EAT_{h2}\ ..\ EAT_{hp}))$$

where l, m, h are the three expertise levels low, medium, and high respectively, and k, r, and p are the corresponding number of designers in each level.

The main idea of this algorithm is to extend the current schedule by one of the steps in the ready list. The tasks in the ready list can be seen as independent tasks if we can define an earliest start time and a deadline for each of them. This is done for the deadlines by propagating them from the terminal to the root nodes in the dependency graph.

The propagated deadline d'(Si) of a step Si is defined by:

1) $d'(Si) = d(Si)$ if $\neg\exists$ Sj : Si precedes Sj

or

2) $d'(Si) = \min \{d(Si), d'(Sj) - tp(Sj)\}$ $\forall$ Sj : Si precedes Sj

In 2) above, if there exists some step Sj such that Si precedes Sj then Sj cannot start until Si has completed. In order to complete Sj's computation before its deadline, the latest time by which Sj must be started is d'(Sj) - tp (Sj). Then Si's real deadline should be d'(Sj) - tp (Sj) if it is smaller than d (Si).

As for the earliest start time (EST) of each step, it is adjusted according to the following:

1) EST' (Si) = EST (Si) if $\neg\exists$ Sj : Sj precedes Si

or

2) EST' (Si) = max {EST (Si), EST'(Sj) + tp (Sj)} $\forall$ Sj : Sj precedes Si

In 2) above, if there exists some step Sj such that Sj precedes Si then Si cannot start until Sj has completed. Since the earliest time that Sj can be completed is EST'(Sj) + tp (Sj) then Si's real EST should be EST'(Sj) + tp (Sj) if it is greater than EST (Si).

The reason for having a pending_list and a ready_list instead of having one ready_list is to give the available tasks (in_degree = 0 and EST <= current time) a fair chance to compete for available designers especially when using different heuristics other than EST first, since the scheduler considers only the steps in the ready_list.

Our scheduling algorithm has two different initialization procedures. The first one is used when the system starts from scratch (i.e., the schedule is empty), while the second initialization procedure is used when new tasks arrive at the system or some of the attributes of an existing step is changed. This scheduling algorithm is similar to the branch and bound technique. The strong feasibility check done before extending the schedule by another node in the search tree is used instead of the lower bound check, normally used with branch and bound algorithm, to bound the search in a given search path. The algorithm works as follows:

Initialization_part:

(1) if initial_schedule = empty

(2) then

initialize EAT values to T0, and the schedule to empty.

perform a Depth First Search on the dependency graph to:

- initialize the in_degree for each node (number of immediate predecessors),
- propagate deadlines, and
- initialize the ESTs (earliest start time) of the steps that have no EST to T0.

insert each pending step (its in_degree = 0) into the pending list ordered by its EST.

(3) else

update the dependency_graph:

- Remove the assigned steps and their corresponding arcs from the dependency graph

- Add the newly arrived steps to the dependency graph (if there is any) checking for the "acyclic" property of the graph and the compatibility of the newly added steps' priorities with that of their successors and predecessors and warn the manager of any violation

Recalculate the in_degree of the graph nodes.

Re-initialize the EAT vector (matrix) to the finish time of the step assigned to each designer and to t for the free designers.

Insert each pending step (its in_degree = 0) into the pending list ordered by its EST.

end if

Schedule_part:

(4) While full_schedule is not reached loop

(5) For all the steps in the pending list:

if EST (S) <= min(EAT) of the corresponding designers then

insert S into the ready_list in order of non-decreasing values of the H (heuristic) function used and delete S from the pending list.

(6) end for

(7) While ready_list is not empty loop

(8) if not STRONGLY_FEASIBLE to extend the schedule by each of the steps in the ready_list then

if the backtrack limit is not reached then increment backtrack counter and backtrack (discard the current partial schedule and backtrack to the previous partial schedule and extend it by a different step)

else exit (NO_FEASIBLE_SCHEDULE)

end if

end if

(9) extend the schedule by the step S that has min H

in case of ties, select the step Si with the highest priority, then the step with max tp(Si)

(10) update the EAT of the assigned designer

(11) update the EST of the immediate successors of S

(12) decrement the in_degree of the immediate successors of S

(13) if the in_degree of any of the immediate successors of S = 0

then

insert it into the pending_list in order of its EST,

end if.

(14) delete S from the ready_list

(15) end while
(16) end while

The STRONGLY_FEASIBLE is a boolean function that works as follows:

FEASIBLE = TRUE
for all the steps S in the ready_list loop
    if min (EAT) of the designers of the same or higher expertise level than
    level(S) + Estimated_duration(S) > deadline(S)
    then FEASIBLE = FALSE
    end if
end for

The following are some of the heuristics that may be used with this algorithm:

- Minimum deadline first (Min_d): H(S) = d (S)
- Minimum earliest start time first (Min_est): H(S) = EST (S)
- Minimum laxity first (Min_L): H(S) = d (S) - (EST (S) + tp (S))
- Min_d + Min_est first: H(S) = W * d (S) + (1-W) * EST (S)
- In the four cases ties are broken using the priorities of the steps (the highest priority step starts first). Further ties are broken by selecting the step that has the maximum tp.

The first three heuristics are simple heuristics and the last one is an integrated heuristic. The weight W ($0 <= W <= 1$), used to combine the two simple heuristics Min_d and Min_est, can be tuned according to the criticalness of the deadlines of the available steps. This means if the deadlines are not critical then W can be set to 0 which leads to Min_est heuristic that is the best for team work to assign tasks to designers according to their earliest start time making a full use of the human resources. On the other hand the value of W can be chosen to favor the deadline heuristic or some way in between to meet the critical deadlines and make the best use of the human resources (designers) available.

The backtracking limit is left open in the cases where the number of tasks is relatively small, and is limited otherwise. In the cases where no feasible schedule is reached either due to the absence of a feasible schedule for the given set of tasks or due to reaching the backtracking limit of the algorithm without reaching one, an algorithm for adjusting the

deadlines is used. This enhancement to the algorithm is presented in section c. This valid schedule can be improved on by using the simulated annealing technique described in section d.

c. Algorithm for Adjusting Deadlines

A valid schedule is a schedule that satisfies the precedence constraints of its tasks but allows some of the tasks to miss its deadlines. Different heuristics can be used to guide the search process to a plausible path that minimizes the number of tasks that must miss its deadlines and in the mean time supports team work by scheduling every available task as soon as the earliest available time of the task is reached. This in turn minimizes the time a designer has to wait for a task to be assigned to him/her.

This algorithm uses almost the same steps as in the previous search algorithm uses with two main differences. The first difference is that: there is one ready_lists for each of the L expertise levels. The main reason for having the different levels of ready_lists is to guarantee that no lower task is assigned to a higher level designer while there is a task of the designer's level ready to be assigned (recall the requirement that the expertise level of the designer must be at least the same as that of the assigned task). The second difference is that when failing the strong feasibility check for extending the schedule by another task, a new deadline is suggested for the task that does not meet its deadline (equal to its calculated finish time). Upon accepting this value by the manager the schedule is extended to the next level and so on until a valid schedule is reached.

The Proposed deadline-adjusting scheduling algorithm works as follows:

initialization_part:

(1) if initial_schedule = empty
(2) then
    initialize EAT values to T0, and the schedule to empty.
    perform a Depth First Search on the dependency graph to:
        - initialize the in_degree for each node (number of immediate predecessors),
        - propagate deadlines, and
        - initialize the ESTs (earliest start time) of the steps that have no EST to T0.

Insert each pending step (its in_degree = 0) into the pending list according to its EST.

(3) else

update the dependency_graph:

- Remove the assigned steps and their corresponding arcs from the dependency graph.

- Add the newly arrived steps to the dependency graph (if there is any) checking for the "acyclic" property of the graph and the compatibility of the newly added steps' priorities with that of their successors and predecessors and warn the manager of any violation.

Recalculate the in_degree of the graph nodes.

Re-initialize the EAT vector (matrix) to the finish time of the step assigned to each designer and to t for those free designers.

Insert each pending step (its in_degree = 0) into the pending list ordered by its EST.

end if

schedule_part:

(4) While full_schedule is not reached loop

(5) For all the steps in the pending list:

if EST (S) <= min(EAT) of the corresponding designers then

insert the step into the corresponding ready_list according to the H (heuristic) function used and delete it from the pending list.

end if

(6) end for

(7) **For all ready_lists from higher_level to lower_level loop**

(8) While ready_list is not empty loop

(9) if not FEASIBLE to extend the schedule by any of the steps in the ready_list

then **suggest a new deadline for the infeasible step assignment**

if the suggestion is not accepted then exit, end if.

end if

(10) extend the schedule by the step S that has min H

(11) update the EAT of the assigned designer

(12) update the EST of the immediate successors of S

(13) decrement the in_degree of the immediate successors of S

(14) if the in_degree of any of the immediate successors of S = 0

then

insert it into the pending_list,

```
                    end if.
(15)                delete S from the ready_list
                    T0 = min (EAT) of the designers of the same or higher expertise
                        level than level(ready_list)
(16)                for all the steps S in the pending list such that expertise_level (S) =
                        level (ready_list):
                        if EST (S) <= T0
                        then
                            insert S into the ready_list according to the H function used
                            and delete it from the pending list.
                        end if
                    end for
                end while
                if not FEASIBLE then exit end if
(18) end for
                if not FEASIBLE then exit end if
(19) end while
```

This algorithm has the property that a designer will never be left idle when there is a ready step that the designer is qualified to do. This is because inserting steps into ready list and their assignment to designers are triggered by the availability of designers as is the case in statement 5, 10, and 15.

As an example to illustrate how this algorithm works, assume we have the same example discussed in the typical scenario in chapter 3 as represented in Table 10. and the corresponding dependency graph in Figure 26. The resources in this example are three designers d1, d2, d3 with expertise levels H, M, L respectively.

TABLE 10. THE STEPS TO BE SCHEDULED AND THEIR ATTRIBUTES

step #	**1**	**2**	**3**	**4**	**5**	**6**
est(S)						
tp(S)	7	6	3	6	5	4
d (S)	18	18	18	22	22	20
predecessor (S)		1	1		2,4	
Successors (S)	2,3	5		5		
r (S)	M	M	M	H	H	L

Now we follow the algorithm step by step to see how it works:

(1) The EAT vector is initialized to zeros EAT = {0, 0, 0} for d1, d2, d3 respectively.

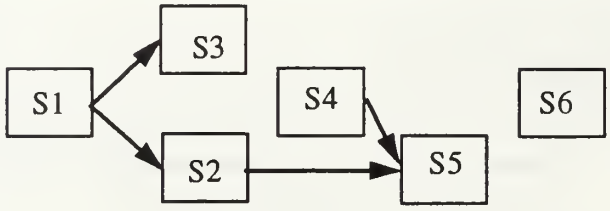


FIGURE 26. The dependency graph

(2) Table 11 shows the new deadlines after propagating them from the terminal nodes all the way up to the corresponding root nodes and initializing the in_degree and the EST.

TABLE 11. THE STEPS ATTRIBUTES AFTER PROPAGATING THE DEADLINES.

step #	1	2	3	4	5	6
est(S)	0	0	0	0	0	0
tp(S)	7	6	3	6	5	4
d (S)	11	17	18	17	22	20
predecessor (S)		1	1		2,4	
Successors (S)	2,3	5		5		
r (S)	M	M	M	H	H	L
in_degree (S)	0	1	1	0	2	0

Using H = d (S)

(3) The steps with in_degree= 0 and EST = 0 are inserted in the ready_list. Now the ready_list = {S1, S4, S6} according to their deadline values.

(4) since the initial partial schedule is empty, it is feasible to schedule any of the three steps in the ready_list without missing any of their deadlines.

(5) The partial schedule is extended by S1 (inserting the steps in a non-decreasing order of their deadlines into the ready_list makes no need to apply the H function, since the step on top of the queue has the min deadline), and S1 is removed from the ready_list. The EAT is updated: EAT = {0, 7, 0}, the in_degree of steps S2, S3 is decremented to 0. The EST(S2) and EST (S3) are set to 7.

(6) Since a feasible full schedule is not reached yet we loop back to step 3. Table 12 reflects the changes after first iteration.

(7.3) S2, S3 have their in_degree = 0, but their EST > min (EAT). Now the ready_list = {S4, S6}

(7.4) The partial schedule is strongly feasible if extended by any of the steps in the ready_list

(7.5) The partial schedule is extended by S4, it is removed from the ready_list, and EAT is updated EAT = {6, 7, 0}, the in_degree of S5 is decremented, and the EST (S5) is set to 6.

TABLE 12. THE EFFECT OF THE FIRST ITERATION

step #	1	2	3	4	5	6
est(S)	0	7	7	0		0
tp(S)	7	6	3	6	5	4
d (S)	11	17	18	17	22	20
predecessor (S)		1	1		2,4	
Successors (S)	2,3	5		5		
r (S)	M	M	M	H	H	L
in_degree (S)	0	0	0	0	2	0

(7.6) Since a full feasible schedule is not reached yet we loop back to step 3. Table 13 reflects the changes after second iteration.

TABLE 13. THE EFFECT OF THE SECOND ITERATION

step #	1	2	3	4	5	6
est(S)	0	7	7	0	6	0
tp(S)	7	6	3	6	5	4
d (S)	11	17	18	17	22	20
predecessor (S)		1	1		2,4	
Successors (S)	2,3	5		5		
r (S)	M	M	M	H	H	L
in_degree (S)	0	0	0	0	1	0

(8.3) no new ready steps yet. The ready_list = {S6}

(8.4) The partial schedule is strongly feasible if extended by any of the steps in the ready_list

(8.5) The partial schedule is extended by S6, it is removed from the ready_list, and EAT is updated EAT = {6, 7, 4}.

(8.6) Since a full feasible schedule is not reached yet we loop back to step 3. Table 14 reflects the changes after third iteration.

(9.3) Now the ready_list is empty, and no full schedule is reached, the time is advanced to max(min (EAT), min (EST (pending_list)))= 7, both S2 and S3 are ready and inserted into the ready list according to their deadline values. Now the ready_list = {S2, S3}

(9.4) The partial schedule is strongly feasible if extended by any of the steps in the ready_list

(9.5) The partial schedule is extended by S2, it is removed from the ready_list, and EAT is updated EAT = {6, 13, 4}. The in_degree of S5 is decremented to 0, and its EST is updated to 13.

TABLE 14. THE EFFECT OF THE THIRD ITERATION

step #	1	2	3	4	5	6
est(S)	0	7	7	0	6	0
tp(S)	7	6	3	6	5	4
d (S)	11	17	18	17	22	20
predecessor (S)		1	1		2,4	
Successors (S)	2,3	5		5		
r (S)	M	M	M	H	H	L
in_degree (S)	0	0	0	0	1	0

TABLE 15. THE EFFECT OF THE FOURTH ITERATION

step #	1	2	3	4	5	6
est(S)	0	7	7	0	6	0
tp(S)	7	6	3	6	5	4
d (S)	11	17	18	17	22	20
predecessor (S)		1	1		2,4	
Successors (S)	2,3	5		5		
r (S)	M	M	M	H	H	L
in_degree (S)	0	0	0	0	1	0

(9.6) Since a full feasible schedule is not reached yet we loop back to step 3. Table 16 reflects the changes after fourth iteration.

(10.3) no new steps to be inserted into the ready_list. Now the ready_list = {S3}

(10.4) The partial schedule is strongly feasible if extended by any of the steps in the ready_list

TABLE 16. THE EFFECT OF THE FIFTH ITERATION

step #	1	2	3	4	5	6
est(S)	0	7	7	0	13	0
tp(S)	7	6	3	6	5	4
d (S)	14	13	10	6	22	20
predecessor (S)		1	1		2,4	
Successors (S)	2,3	4		5		
r (S)	M	M	M	H	H	L
in_degree (S)	0	0	0	0	0	0

(10.5) The partial schedule is extended by S3, it is removed from the ready_list, and EAT is updated EAT = {10, 13, 4}.

(10.6) Since a full feasible schedule is not reached yet we loop back to step 3. Table 16 reflects the changes after fifth iteration.

(11.3) Again the ready_list is empty, the time is advanced to 10 then to 13 where S5 becomes ready and inserted into the ready_list. Now the ready_list = {S5}

(11.4) The partial schedule is strongly feasible if extended by any of the steps in the ready_list

(11.5) The partial schedule is extended by S5, it is removed from the ready_list, and EAT is updated EAT = {18, 13, 4}.

(11.6) Since a full feasible schedule is reached the algorithm stops. The resulting schedule is shown in Table 17.

TABLE 17. THE RESULTING SCHEDULE

step #	designer	start_time	finish_time
S1	d2	T0	T0+7
S2	d2	T0+7	T0+13
S3	d1	T0+7	T0+10
S4	d1	T0	T0+6
S5	d1	T0+13	T0+18
S6	d3	T0	T0+4

V. EVALUATION AND VALIDATION

A. COMPLEXITY ANALYSIS

Both of the two algorithms introduced in Chapter IV.C.1.b and IV.C.1.c have a total of n steps, where n is the number of the tasks to be scheduled. The complexity of each step is determined by the complexity of the computation done to determine strong feasibility and the complexity of H function evaluation. The strong feasibility calculation is linearly proportional to the number of the steps in the ready list. This number depends on the connectivity of the dependency graph which is n in the worst case. The H function computation is done simply by inserting the ready steps into the ready list(s) in order of their H function which has the order of (log n) in the worst case if we use a heap data structure for the ready lists.

The overall worst case complexity of the algorithm is:

$$n + (n - 1) + (n - 2) + .. +2 = O(n^2).$$

The backtracking in of the algorithm in Chapter IV.C.1.b can be limited to a constant number which does not affect the complexity analysis. In our expermintal results we found out that the number of backtracking is at most proportional to n with a small constant (0.75). It is also worth noting that the number of steps in the ready_list is linearly proportional to the remaining ready unassigned steps which is always less than or equal to the number of the remaining unassigned steps.

B. Simulation Study

The main goal of a scheduling algorithm is to find a feasible schedule for a set of tasks, if one exists. Clearly, a heuristic scheduling algorithm is not guaranteed to reach such a schedule. However, one heuristic algorithm is favored over another, if we have a number of task sets that known to have feasible schedules, the first is able to find feasible schedules for more task sets than the second. To take this approach, we have to come up with a number of task sets, each of which is known to have a feasible schedule. Unfortunately, only an exhaustive algorithm can find out whether an arbitrary task set can be feasibly scheduled.

Given m different designers, the complexity of an exhaustive search to find a feasible schedule for n tasks in the worst case can be $O(m^n * n!)$. This is why we take the approach taken by Ramamritham et. al. [Ref. 67] which is using a task generator that can generate schedulable task sets where the number of tasks in each set can be arbitrarily large without adding much complexity on the task generator. Additionally, the tasks are generated to guarantee the total utilization of the available designers. These task sets are then input to the scheduling algorithm, that has no knowledge that these sets are schedulable. The parameters used to generate task sets are:

1. The minimum duration of a task, Min_D.
2. The maximum duration of a task, Max_D.
3. The schedule length, L.

The task set generator starts with an empty EAT matrix, it then generates a task by selecting one of the n designers that have the earliest available time and then randomly chooses the task duration between the minimum duration and the maximum duration. The task generator then increments the EAT of the selected designer by the value of the task duration. The task generator generates tasks until the remaining unused time units of each designer, up to the schedule length L, is less than the minimum duration of a task, that means no more tasks can be generated for this designer within the given schedule length.

The deadline for each task is chosen randomly between the task 's shortest completion time T_{sc} and $(1+F) * T_{sc}$, where F is a parameter indicating the tightness of the deadlines, and is related to the loading factor of each set of designers of the same expertise level. If F is 0, the scheduler must be able to find the same schedule as that found by the task generator in order to reach a feasible schedule. As the value of F is increased it is obvious that the scheduler has a better chance to find a feasible schedule for the task set.

1. Simulation Method

In our simulation study, N task sets are generated, where each set is known to be schedulable according to the task set generation procedure discussed above. Performance of different heuristics are compared according to how many of the N feasible task sets are

found schedulable when the heuristics are used [Ref. 67]. We use the same metric used in [Ref. 67] which is defined as:

$SR = \frac{s}{N}$, where s is the number of schedulable task sets found by the heuristic algorithm, and N is the total number of task sets.

The loading factor for the designers is different according to their expertise level. we assume that the designers are of three different expertise levels High, medium and low, and a step can be assigned to a designer that has at least the same expertise level as that required by the step. This assumption make the loading factor varies for the designers in different levels as defined below.

For high level designers we define the loading factor as follows:

$$LFh = \frac{\sum Tph}{(Max(di) - To) \times Mh}$$

where LFh is the loading factor for high level designers, Tph is the estimated duration for a high level task, To is the initial start time for scheduling the tasks, Mh is the number of available high level designers and di is the deadline of task i.

For a medium level designer we define the loading factor as follows:

$$LFm = \frac{\sum Tpm}{(Max(di) - To) \times (Nm + Nh - Nh \times LFh)}$$

$$LFm = \frac{\sum Tpm}{(Max(di) - To) \times Mm + (1 - LFh)\ (Max(di) - To) \times Mh}$$

where LFm is the loading factor for medium level designers, Tpm is the estimated duration for a medium level task and Mm is the number of available medium level designers.

For a low level designer we define the loading factor as follows:

$$LFl \frac{\sum Tpl}{(Max(di) - To) \times Nl + (1 - LFm)(Max(di) - To) \times (Nm + Nh - Nh \times LFh)}$$

$$LFl = \frac{\sum Tpl}{(Max(di) - To)(N - Nh(LFh + LFm - LFh \times LFm) - LFm \times Nm)}$$

where LFl is the loading factor for low level designers, Tpl is the estimated duration for a low level task and Ml is the number of available low level designers.

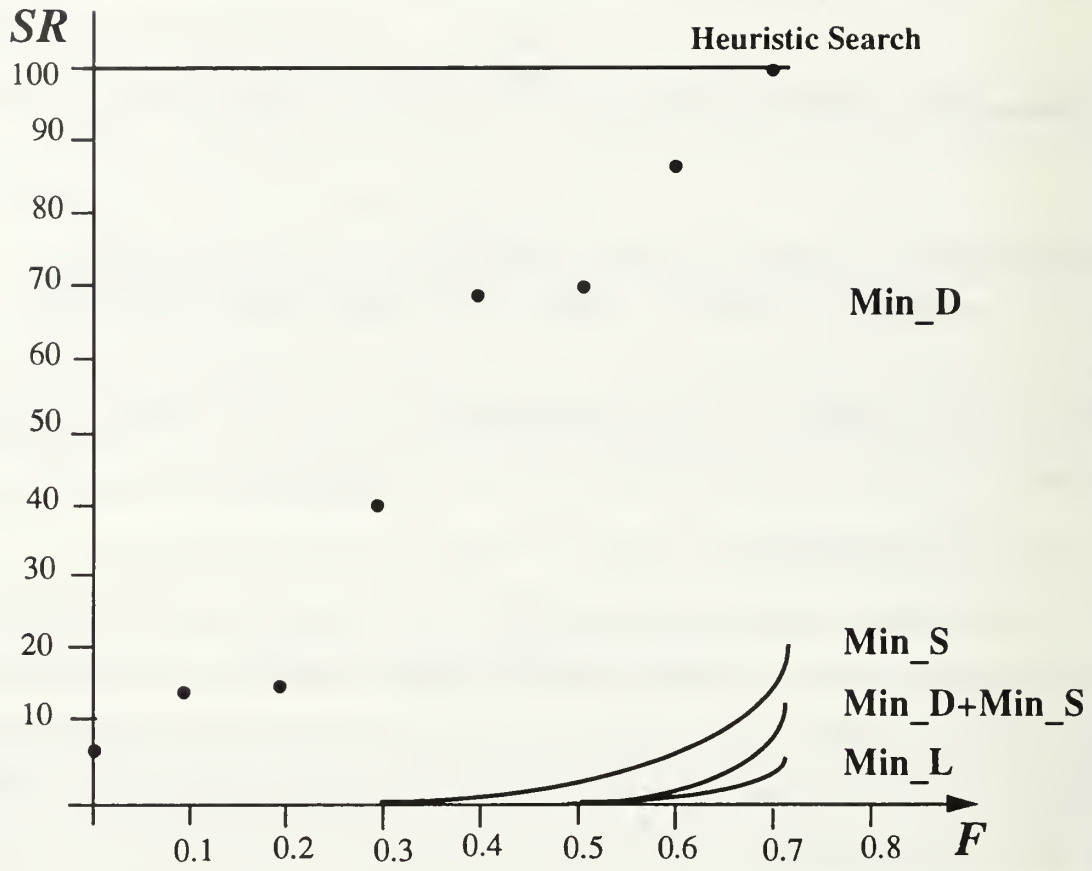
2. Simulation Results

The system, in our experiment, consists of three designers, one of each expertise level high, medium and low. Tasks durations are randomly chosen between Min_D (2) and Max_D (20). The number of task sets generated is 50, and each task set has between 28 and 31 tasks. We present the results as shown in Table 18, and in plot form in Figure 27 where the success ratio SR is plotted on the Y-axis and F on the X-axis (F is related to laxity). Simulation parameter is F to measure the sensitivity of each heuristic algorithm to the change in laxities.

TABLE 18. Relation between Success Ratio (SR) and Laxity (L)

Laxity (F)	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
Heuristic Search	100	100	100	100	100	100	100	100
Min_D	6	14	14	40	70	72	86	100
Min_s	0	0	0	0	8	10	10	22
Min_D + Min_S	0	0	0	0	0	0	8	16
Min_L	0	0	0	0	0	0	8	10

As can be seen from Figure 27 the greedy heuristics Min_D, Min_S, Min_D+ Min_S and Min_L perform poorly due to the dependency relations between the tasks. We found that the heuristic search algorithm have a success ratio of 100% even when the deadlines are very tight (F=0). It is worth noting that this excellent performance by the



heuristic search algorithm is obtained with unlimited backtracking. This leads us to study the effect of limiting the backtracking.

Instead of trying different backtracking limits and studying their effects on the performance of the algorithm, we do it the other way around by counting how many times the algorithm backtracks to get a feasible schedule given the different task sets. The results is shown in Table 19 where the number of backtracking is represented as a percentage of the total number of tasks in a task set. The results plotted in Figure 28 shows that the backtracking limit in the worst case (tightest deadlines: F=0) is approximatly 0.6 N, where N is the number of tasks in a task set, and this limit decreases significantly as the deadlines are relaxed.

FIGURE 27. Relation between Success Ratio (SR) and Laxity (L)

TABLE 19. NUMBER OF BACKTRACKING (AS PERCENTAGE OF N) AND LAXITY (L)

Laxity (F)	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
backtracking #	.57	.27	.16	.075	.034	.012	0.0	0.0

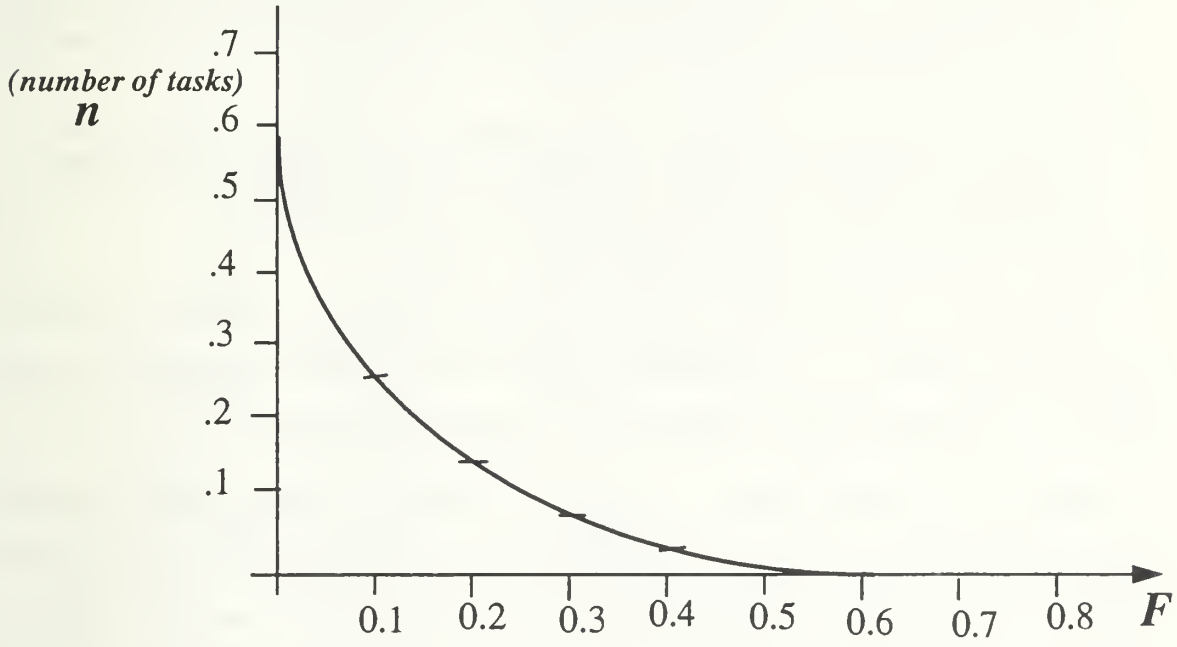


FIGURE 28. Limiting Backtracking

C. DEVELOPMENT OF TEST CASES

To evaluate the ECS system and its proposed functionality, our test cases are designed to show that the ECS realizes the two main claims; 1) The ECS provides automated support for changes in the plan during the execution of the plan. 2) It provides automatic decision support for planning and team coordination. We show that the ECS system realizes these two main issues by tailoring the test cases to answer a set of questions showing the fine details pertaining to both of these issues. The set of questions are as follows:

1. Does the ECS support incremental planning of new evolution steps as they become available?

2. Does the ECS respond on the fly to changes in the plan (the attributes of the existing steps and the changes in the design team members) reflecting their effects on the current plan (schedule)?
3. Does the ECS automatically determine change consequences? (calculates the affected modules (components) by each change (step) needed for propagating the change consequences as well as calculating the set of secondary inputs needed to perform the required change)
4. Does the ECS automatically guarantee the consistency of the project database? (create a substep for each affected module by the proposed top step and include it in the plan for implementing the change and restrict the commitment of the top step by the completion of all its substeps)
5. Does the ECS support automatic VCCM? (determines and saves the version and variation numbers of the outputs of committed steps and generates a new system configuration at the commitment of each top level evolution step?)
6. Does the ECS support parallel multisystem evolution?

In this section we run similar scenarios to those introduced in Chapter III.F as our test cases for the evaluation and validation of the ECS system performance. During the test cases we indicate the answers to the different questions presented above.

The systems we are evolving are called "c3i_system" and "fishies" where simplified block diagrams of their decomposition are shown in Figure 29 and Figure 30 to make it easy to follow the different cases. The C3i_system is a Command, Control and Communication information system developed at the CAPS lab. The fishies system is fish farm control system also developed in the CAPS lab. Notice that thick arrows indicate both "part_of" and "used_by" relationships and the thin arrows indicate only the "used_by" relationship among the system components.

The set of initial designers are: "badr" with expertise_level "low", "brockett" with expertise_level "medium" and "dampier" with expertise_level "high".

In the following sub-sections we follow a typical scenario for evolving the two systems mentioned above indicating in each subsection the ECS system features demonstrated.

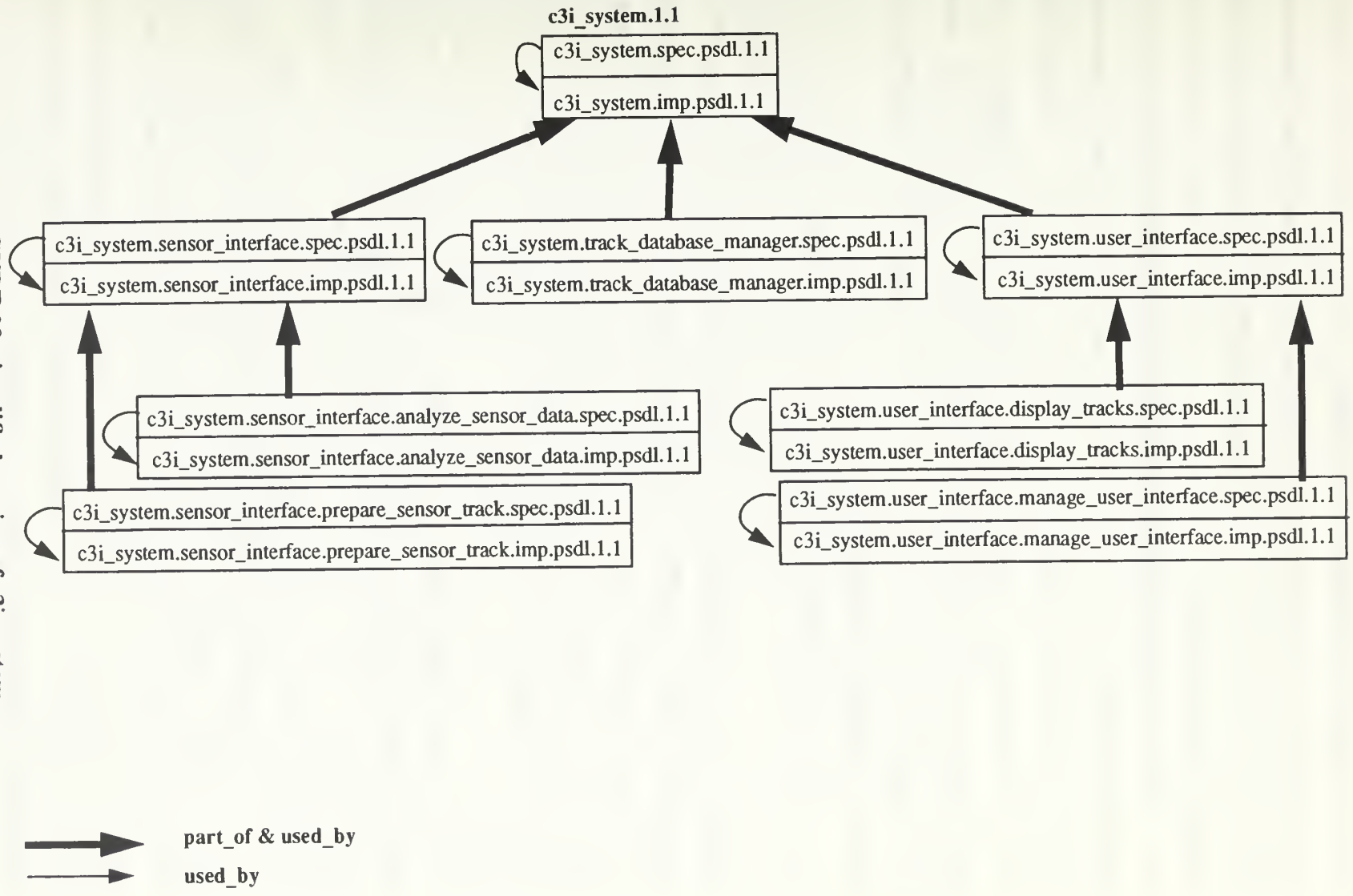


FIGURE 29. simplified version of c3i_system

1. Determining Change Consequences

In this section we show how the creation of a step with a given primary input automatically generates the modules affected by this step as well as the secondary inputs needed to perform this step.

1. Starting with the initial configuration of "c3i_system 1:1" (variation number 1 and version number 1), we create three steps: first step with primary input "c3i_system.sensor_interface.spec.psdl", second step with primary input "c3i_system.spec.psdl", third step with the primary input "c3i_system.user_interface.manage_user_interface.imp.psdl". As soon as the manager clicks the apply button for creating a step the ECS assigns a unique number to it and generates the affected modules of this step.

As a result of creating first step, the ECS assigns it a unique number, 1 in this case, and generates the modules affected by this step which are "c3i_system.imp.psdl and c3i_system.sensor_interface.imp.psdl". The result of creating the second step is assigning it the unique number 2 as its step_id and generating the affected module by the step which is "c3i_system.imp.psdl". Finally the result of creating the third step is assigning it the unique number 3 as its step_id and generating its affected modules which are "none" in this case because implementation modules do not affect any other modules and the ECS also generates its secondary input which is "c3i_system.user_interface.manage_user_interface.spec.psdl". Notice that secondary inputs generated in the first two steps were none because the primary inputs in both cases were specification modules that have no secondary inputs yet since we did not have the requirement modules in our database yet (planned to be added in our future work).

2. Two more steps are created starting with initial version of "fishies 1:1" (the fish farm control system). The first step with primary input "fishies.Control_water_Flow.spec.psdl" is automatically assigned the number 4 and the two modules "fishies.imp.psdl and fishies.Control_water_Flow.imp.psdl" are generated as its affected modules. The second step with primary input "fishies.Display_Status.imp.psdl" is automatically assigned the number 5 and generates no affected modules and the ECS

generates "fishies.Display_Status.imp.psdl" as its secondary input. Images of the ECS screens for the details of the created steps and their automatically generated attributes (affected modules and secondary inputs) are presented in Appendix D Figure 35, Figure 36, Figure 37, Figure 38, and Figure 39.

Notice that: Creating a step automatically generates the affected modules by the step as well as the secondary inputs used by the primary input of the step. (answering question 3).

2. Enforcing Change Consequences for Global Consistency

In this subsection we show how the ECS, after the manager approval of a step, will automatically create a substep of the approved step for each affected module by the step. This is to guarantee that all the change consequences are considered as part of the original change and, as we will show later, that completing all the substeps of top level step is enforced as a condition for the completion of the top step.

The manager reviews the created steps, adds the deadline, priority, and expertise level for each step, then he decides to approve steps 1, 2 and 4. The result of approving step 1 is creating three substeps with the unique numbers 6, 7, and 8. Step 6 has the primary input "c3i_system.sensor_interface.spec.psdl" which is the primary input of step 1, that is because step 1 is now a composite step and will only be used for controlling and enforcing the completion of its three substeps as a condition for its completion. Step 7 has the primary inputs "c3i_system.imp.psdl" and its secondary inputs are automatically generated (the spec component of c3i_system and the spec components of its three children as shown in Figure 42 in Appendix D). Step 8 has the primary input "c3i_system.sensor_interface.imp.psdl" and it has the spec of the same component and the specs of its two children as its secondary inputs as shown in Figure 43 in Appendix D). Both of steps 7 and 8 automatically have step 6 as its predecessor, since the results of step 6 are

the basis for the changes required by them (the output of step 6 is a secondary input for both steps 7 and 8).

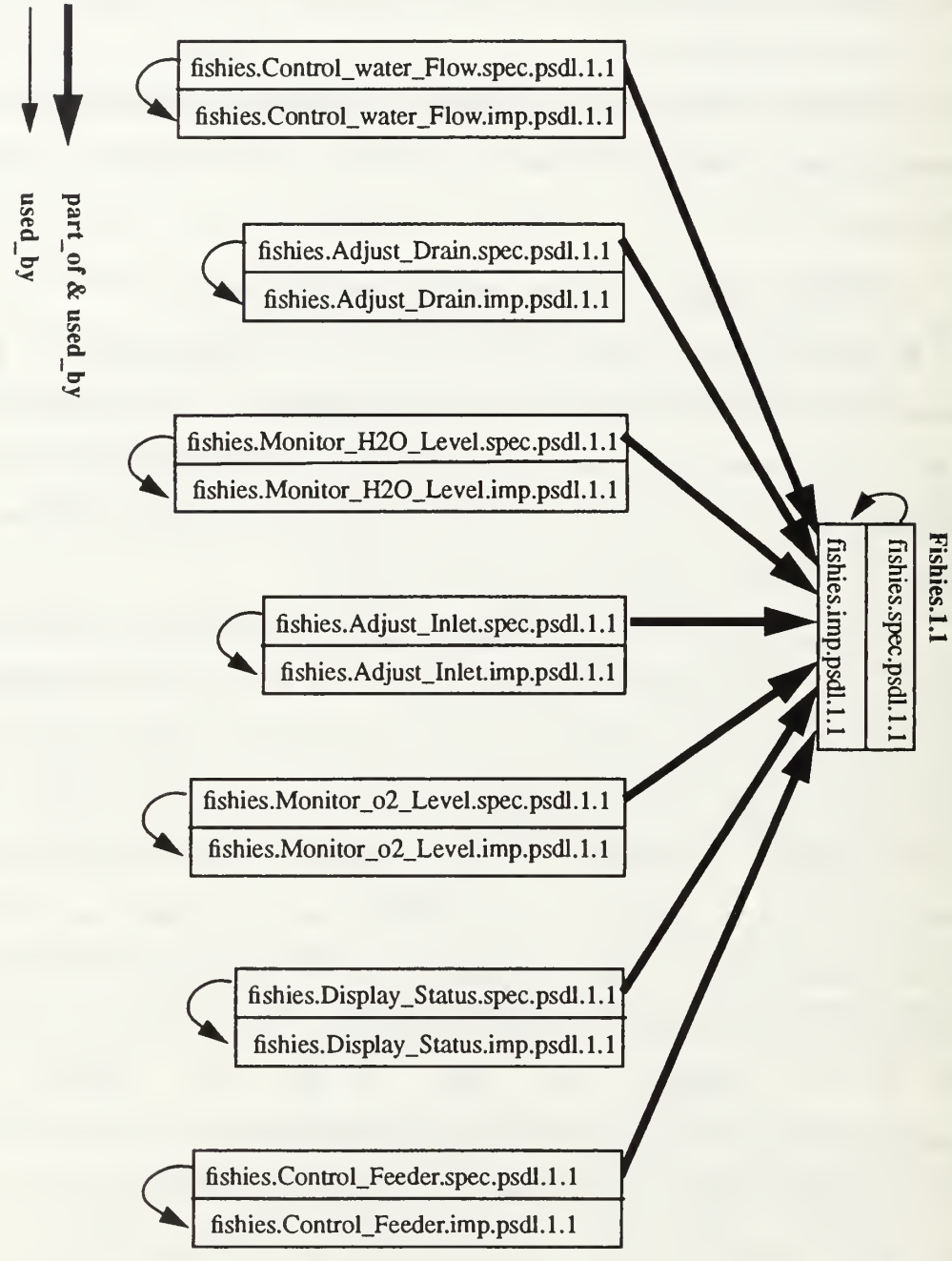


FIGURE 30. simplified version of the fish-farm system (fishies)

Approving step 2 automatically creates two substeps 9 and 10 with primary inputs "c3i_system.spec.psdl" and "c3i_system.imp.psdl" respectively where step 9 precedes step

10. Approving step 4 also creates three substeps 11, 12, and 13 with the primary inputs and secondary inputs shown in Figure 48, Figure 49, and Figure 50 in Appendix D. Step 11 precedes both of steps 12 and 13 as its the basis for their changes.

It is worth noting that all the substeps inherit the deadlines, expertise_level and the priorities of their top level steps as shown in the corresponding figures of the screen images of the details of the steps in Appendix D.

Notice that: Approving a step creates a substep for each affected module by the proposed top step. The newly created steps are atomic and inherits the status "approved" of their top steps besides their attributes (deadlines, priorities, and expertise_level). (answer for question 4).

3. Incremental Planning

1. The manager enters an estimated duration for the substep from 6 -13 using the edit_step command. The values entered are 6, 4, 5, 7, 3, 6, 5, and 4 hours respectively.

2. The manager starts scheduling the steps starting with step 1(related to c3i_system project). Clicking on "schedule step" option in the manager's menu and entering the step number 1, the system outputs the following schedule for the three substeps of step 1:

```
STEP_ID   S_LEVEL   D_NAME      START_TIME   FINISH_TIME
-------   -------   ----------  ----------   -----------
   6      MEDIUM    brockett        0            6
   8      MEDIUM    dampier         6            10
   7      MEDIUM    brockett        6            11
Confirmation required to save the schedule in DDB. Answer(Y/N) : y
```

3. As soon as the manager confirms the output of the scheduler, the ECS saves it in the design database replacing the hours time units for start_time and finish_time with real dates and time of day, as captured from the show_schedule screen below:

6	11/06/93 08:32	11/06/93 14:32	brockett
8	11/06/93 14:32	11/07/93 10:32	dampier
7	11/06/93 14:32	11/07/93 11:32	brockett

The ECS also sends an E_mail message to brockett informing him about his assignment.

4. Next, the manager schedules step 4 (related to the fish farm project). Clicking on "schedule_step" option in the manager's menu and enters the step number 4, the ECS outputs the following updated schedule.

```
STEP_ID   S_LEVEL  D_NAME      START_TIME  FINISH_TIME
-------   -------  ----------  ----------  -----------
   11     LOW       badr            0           6
   12     LOW       badr            6           11
   7      MEDIUM    dampier         6           11
   8      MEDIUM    brockett        6           10
   13     LOW       brockett        10          14
Confirmation required to save the schedule in DDB. Answer(Y/N): y
```

Notice that step 6 does not show in the output of the scheduler because it is already assigned to a designer and the ECS does not give the scheduler the flexibility to change the assignment for step 6 to save the work done on that step. However step 6 still appears in the collective schedule saved in the design database as shown below.

6	**11/06/93 08:32**	**11/06/93 14:32**	**brockett**
8	**11/06/93 14:32**	**11/07/93 10:32**	**brockett**
7	**11/06/93 14:32**	**11/07/93 11:32**	**dampier**
11	**11/06/93 08:36**	**11/06/93 14:36**	**badr**
12	**11/06/93 14:36**	**11/07/93 11:36**	**badr**
13	**11/07/93 10:36**	**11/07/93 14:36**	**brockett**

One more feature of the scheduler is assigning the ready step 11 to badr right away since he is free at the time the step is scheduled for him and sends him an E_mail informing him about his assignment as shown below.

Notice that the time the E_mail message is sent to designer badr is the same as the start time of his assignment in the schedule, because designer badr is free at the time step 4 is scheduled.

Before scheduling step 2 the manager enters step1 as predecessor of step 2 because their substeps (step 7 and step 10) are modifying the same component. This modification automatically makes step 7 a predecessor of step 10, which prevents the start of step 10 until the completion of step 7.

```
From badr Sat Nov  6 08:36:54 1993
Return-Path: <badr>
Received: from suns7-caps.cs.nps.navy.mil (suns7.cs.nps.navy.mil)
ps.navy.mil (4.1/SMI-4.1)
        id AA06828; Sat, 6 Nov 93 08:36:54 PST
Date: Sat, 6 Nov 93 08:36:54 PST
From: badr (salah badr)
Message-Id: <9311061636.AA06828@taurus.cs.nps.navy.mil>
To: badr
Status: R

You have been assigned the step no:   11
```

Using the same command as in 4 above the manager schedules step 2. The resulting output of the scheduler is shown below as a screen image:

```
STEP_ID   S_LEVEL   D_NAME      START_TIME   FINISH_TIME
-------   -------   ----------  ----------   -----------
   9      HIGH      dampier          0            7
   12     LOW       badr             6            11
   8      MEDIUM    brockett         6            10
   7      MEDIUM    dampier          7            12
   13     LOW       brockett         10           14
   10     HIGH      dampier          12           15
Confirmation required to save the schedule in DDB. Answer(Y/N): y
```

Notice that step 6 and 11 do not appear in the output of the scheduler because they are already assigned, however they still appear in the schedule in the DDB as shown in the screen image of the saved schedule below. Also an e_mail message is sent to designer dampier informing him about his assignment.

6	11/06/93 08:32	11/06/93 14:32	brockett
8	11/06/93 14:32	11/07/93 10:32	brockett
7	11/06/93 15:45	11/07/93 12:45	dampier
11	11/06/93 08:36	11/06/93 14:36	badr
12	11/06/93 14:36	11/07/93 11:36	badr
13	11/07/93 10:36	11/07/93 14:36	brockett
9	11/06/93 08:45	11/06/93 15:45	dampier
10	11/07/93 12:45	11/07/93 15:45	dampier

Notice also that step 10 is scheduled to start after both step 7 and 9 (its predecessors) scheduled finish times.

As a conclusion of this subsection, we have shown that: 1) The ECS incrementally schedules the steps as soon as the manager decides to do so. 2) Preserves the precedence between the steps and meeting their deadlines when it is feasible to do so. 3) Ensuring there is no the case that a designer is idle and there is a ready step he can do and not assigned to him. 4) automatically informs each designer of his due assignment.

4. Changes in the Plan

Possible changes in the plan include the changes in the attributes of existing active steps such as deadlines, priority, precedence, estimated duration or even adjusting the affected modules and secondary input as well as suspending or abandoning any of the steps which may affect the plan (schedule). Another possible changes are the changes in the designer pool by adding, deleting or changing the expertise level of a designer. As an example of these changes we will examine changing the estimated duration of some of the steps either by an early commit of the step or increasing the estimated duration of another step. We will also examine the effects of suspending a step and deleting a designer.

a. Early Commit of a Step

Now designer brockett finishes his work on step 6 earlier than planned (for the sake of the test example) and commits his step. His commit command saves his changes to the design database creating new version of its input and automatically assigns it a version and variation numbers (variation 1 and version 2 in this case). This also triggers the scheduling mechanism to find his next assignment and adjust the schedule accordingly as shown in the screen image of the updated schedule below.

8	11/06/93 09:46	11/06/93 13:46	brockett
7	11/06/93 13:46	11/07/93 10:46	brockett
11	11/06/93 08:36	11/06/93 14:36	badr
12	11/06/93 14:36	11/07/93 11:36	badr
13	11/06/93 15:46	11/07/93 11:46	dampier
9	11/06/93 08:45	11/06/93 15:45	dampier
10	11/07/93 11:46	11/07/93 14:46	dampier

As shown in the schedule above step 6 is removed from the current schedule. Designer brockett gets his next assignment which is step 8. The status of step 6 is changed

to "completed", and its finish time is set to 11/6/93 9:46 as shown in the screen image of the details of step 6 in Figure 51 in Appendix D.

Notice that the availability of designer brockett earlier affected which designer does which step for steps that are scheduled but not yet assigned. Now step 7 which was scheduled before for dampier is scheduled to brockett since it requires his expertise level and he should be available to perform this step before its deadline. This also changed the planned assignment for dampier from step 7 and 10 to steps 13 and 10. As a result the scheduled finish times for step 2 and 4 (and their substeps) is improved (become earlier than before committing step 6).

If we look at the status of the different steps in the system after the commitment of step 6 (using the command show_steps with option "all") we notice that step 6 is completed while steps 8, 11, and 9 are assigned which automatically makes the steps 1, 2, and 4 have the assigned status (according to the relations between top steps and their substeps defined in chapter 3 and specified in Appendix A. The rest of the steps (except step 3 and 5) are scheduled with estimated start time as indicated in the schedule. Step 3 and 5 are still in the proposed state.

```
step_set has 13 items.

5,  Status: proposed
6,  Status: completed
7,  Status: scheduled
1,  Status: assigned
9,  Status: assigned
11,  Status: assigned
12,  Status: scheduled
3,  Status: proposed
4,  Status: assigned
8,  Status: assigned
10,  Status: scheduled
2,  Status: assigned
13,  Status: scheduled
```

b. Increasing Estimated Duration of a Step

According to the schedule step 11 which is assigned to designer badr is due to complete at 11/6/93 14:36, but he is asking to extend the estimated duration by 2 more hours. Now the Manager edits the estimated duration of step 11 to be 8 hours instead of 6 using the edit step command from his menu. The result of this change is reflected automatically on the schedule as shown from screen image of the new schedule below.

8	11/06/93 09:46	11/06/93 13:46	brockett
7	11/06/93 13:46	11/07/93 10:46	brockett
11	11/06/93 08:36	11/06/93 16:36	badr
12	11/06/93 16:36	11/07/93 13:36	badr
13	11/06/93 16:36	11/07/93 12:36	dampier
9	11/06/93 08:45	11/06/93 15:45	dampier
10	11/07/93 12:36	11/07/93 15:36	dampier

Notice that the finish time of step 11 is changed to be 16:36 instead of 14:36 in the previous schedule, and accordingly designer badr's next assignment is shifted to start at 16:36.

c. Suspending a Step

Now according to a management decision step 4 has to be suspended because the fish farm owner has a budget problem and cannot afford this change now but he will go with the other change proposed by step 5.

The manager uses the suspend step command to suspend step 4 which automatically takes its substeps (11, 12, and 13) out of the schedule and their status is changed back to approved. The screen image of the updated schedule after suspending step 4 is shown below.

8	11/06/93 09:46	11/06/93 13:46	brockett
7	11/06/93 13:46	11/07/93 10:46	brockett
9	11/06/93 08:45	11/06/93 15:45	dampier
10	11/07/93 10:47	11/07/93 13:47	dampier

because of suspending step 11 (part of step 4), an E_mail message is sent to designer badr informing him that his step is suspended as shown below.

```
From badr Sat Nov  6 11:47:52 1993
Return-Path: <badr>
Received: from suns7-caps.cs.nps.navy.mil (suns7.cs.nps.navy.mil)
ps.navy.mil (4.1/SMI-4.1)
        id AA08050; Sat, 6 Nov 93 11:47:46 PST
Date: Sat, 6 Nov 93 11:47:45 PST
From: badr (salah badr)
Message-Id: <9311061947.AA08050@taurus.cs.nps.navy.mil>
To: badr
Status: R

Your current assigned step:  11      has been Suspended...

&
```

Now the manager approves step 5, which automatically creates one substep with the unique number 14. The manager adds an estimated duration of 6 hours to step 14 using the edit step command, then using schedule step command from his menu schedules step 5 which automatically adds its atomic substep 14 to the schedule and assigns it to designer badr who is idle at this point. The screen image of the updated schedule is shown below.

8	**11/06/93 09:46**	**11/06/93 13:46**	**brockett**
7	**11/06/93 13:46**	**11/07/93 10:46**	**brockett**
9	**11/06/93 08:45**	**11/06/93 15:45**	**dampier**
10	**11/07/93 10:47**	**11/07/93 13:47**	**dampier**
14	**11/06/93 11:57**	**11/07/93 09:57**	**badr**

The ECS also sends an E_mail message to badr informing him about his new assignment as shown below.

The manager also approves step 3 and enters duration of 10 hours to its substep (step 15), then he schedules step 3. The scheduler finds out that the deadline for step 15 cannot be met. It suggests the calculated finish time to be used as the deadline of this step. The screen image of the scheduled is shown below.

Upon the acceptance of the manager to the system suggestion it produces a feasible schedule according to the new deadline and automatically changes the deadline for step 3 and step 15 (the substep of step 4) to the new value as shown in the screen images of

```
From badr Sat Nov  6 11:57:37 1993
Return-Path: <badr>
Received: from suns7-caps.cs.nps.navy.mil (suns7.cs.nps.navy.mil)
ps.navy.mil (4.1/SMI-4.1)
        id AA08078; Sat, 6 Nov 93 11:57:36 PST
Date: Sat, 6 Nov 93 11:57:36 PST
From: badr (salah badr)
Message-Id: <9311061957.AA08078@taurus.cs.nps.navy.mil>
To: badr
Status: R

You have been assigned the step no:    14

&
```

```
in-feasible schedule: step # 15
suggested deadline should be >= 13
Would you like to change it?  Answer(y/n)y

Enter the new Deadline 13
STEP_ID    S_LEVEL   D_NAME       START_TIME   FINISH_TIME
-------    -------   ----------   ----------   -----------
   7       MEDIUM    brockett         1            6
   15      LOW       dampier          3            13
   10      HIGH      dampier          13           16
Confirmation required to save the schedule in DDB. Answer(Y/N):
```

step 3 and step 15 in Figure 55 and Figure 55 in Appendix D. The screen image of the updated schedule is shown below.

8	**11/06/93 09:46**	**11/06/93 13:46**	**brockett**
7	**11/06/93 13:46**	**11/07/93 10:46**	**brockett**
9	**11/06/93 08:45**	**11/06/93 15:45**	**dampier**
10	**11/08/93 09:46**	**11/08/93 12:46**	**dampier**
14	**11/06/93 11:57**	**11/07/93 09:57**	**badr**
15	**11/06/93 15:46**	**11/08/93 09:46**	**dampier**

d. Committing a Step

Now to show the automated VCCM capabilities of the ECS let us commit the substeps of step 1 then step 1.

First let designer brockett commits step 8. This automatically updates the schedule as shown below. This leads to assigning brockett step 7 and sending him an E_mail message informing him about his new assignment.

7	11/06/93 13:46	11/07/93 10:46	brockett
9	11/06/93 08:45	11/06/93 15:45	dampier
10	11/08/93 09:46	11/08/93 12:46	dampier
14	11/06/93 11:57	11/07/93 09:57	badr
15	11/06/93 15:46	11/08/93 09:46	dampier

Now for the sake of the example let designer brockett commits step 7. This is an early commit which automatically updates the schedule as shown below.

9	11/06/93 08:45	11/06/93 15:45	dampier
10	11/06/93 15:52	11/07/93 10:52	dampier
14	11/06/93 11:57	11/07/93 09:57	badr
15	11/06/93 13:52	11/07/93 15:52	brockett

Notice that as soon as designer brockett commits step 7 the system assigns him step 15 which was planned for designer dampier before, because step 15 is ready and designer brockett becomes available after committing step 7.

Before committing step 1 let us have a look at the versions of both c3i_system and fishies prototypes in the database using show prototypes command as shown below.

```
fishies   Has the following versions:
fishies11

c3i_system   Has the following versions:
c3i_system11
```

The manager commits step 1 using commit step command from his menu when all the verification and checking for the substeps are done. The result of this command is creating version number 2 on variation number 1 of the c3i_sysem as shown below.

```
fishies   Has the following versions:
fishies11

c3i_system   Has the following versions:
c3i_system11
c3i_system12
```

Now if we look at the available steps at the system we notice that step 1 and its substeps 6, 7, and 8 are all have the status completed when we use the show steps with the option completed from the manager menu as shown below.

```
step_set has 15 items.

6,      Status: completed
7,      Status: completed
1,      Status: completed
8,      Status: completed
```

The screen images of steps 6, 7, 8 and 1 after they have been completed showing their expertise_level, the designer assigned to each step the start and finish times as well as the rest of the attributes are shown in Figure 51, Figure 53, Figure 55, and Figure 54 in Appendix D.

One more feature of the ECS is related to the default base version to which the top step is applied. When step 1, 2, and 3 are created as top level steps they had the c3i_system 1:1 as the base version for the three steps. When step 1 is committed producing c3i_system 1:2 the default base version for both steps 2 and 3 is automatically changed to be the newly created version c3i_system 1:2 as shown in the screen images of step 2 and 3 in Figure 55 and Figure 55 in Appendix D.

Another important feature of the ECS is the automatic warning to both manager and designer one hour before a step is due to commit as shown in the E-mail message below received by the manager.

e. Dropping a Designer

Designer dampier commits step 9, and the manager decides to schedule step 4 again. Remember that when step 4 was suspended before its status changed back to approved. The updated schedule after committing step 9 is shown below.

```
10      11/06/93 15:52      11/07/93 10:52      dampier
14      11/06/93 11:57      11/07/93 09:57      badr
15      11/06/93 13:52      11/07/93 15:52      brockett
```

```
From badr Sat Nov  6 14:26:18 1993
Return-Path: <badr>
Received: from suns7-caps.cs.nps.navy.mil (suns7.cs.nps.navy.mil)
ps.navy.mil (4.1/SMI-4.1)
        id AA08946; Sat, 6 Nov 93 14:26:18 PST
Date: Sat, 6 Nov 93 14:26:18 PST
From: badr (salah badr)
Message-Id: <9311062226.AA08946@taurus.cs.nps.navy.mil>
To: badr
Status: R

ATTENTION REQUIRED  Step:  9       should commit within an hour...
```

The manager uses schedule step command for step 4 then the ECS produces the updated schedule below.

10	**11/06/93 15:52**	**11/07/93 10:52**	**dampier**
14	**11/06/93 11:57**	**11/07/93 09:57**	**badr**
15	**11/06/93 13:52**	**11/07/93 15:52**	**brockett**
11	**11/07/93 09:57**	**11/08/93 09:57**	**badr**
13	**11/08/93 09:57**	**11/08/93 13:57**	**dampier**
12	**11/08/93 09:57**	**11/08/93 14:57**	**badr**

Now the manager decided to send designer badr to one of the sites, so he must delete him from the schedule. The manager uses drop designer option from the edit_team sub-menu. After the system asks for the manager's confirmation, it suggests deadline changes for both steps 13 and 12 as shown below.

When the suggested deadline changes is accepted by the manager, the ECS produces the following updated schedule.

10	**11/06/93 15:52**	**11/07/93 10:52**	**dampier**
14	**11/07/93 10:59**	**11/08/93 08:59**	**dampier**
15	**11/06/93 13:52**	**11/07/93 15:52**	**brockett**
11	**11/07/93 15:59**	**11/08/93 15:59**	**brockett**
13	**11/08/93 15:59**	**11/09/93 11:59**	**dampier**
12	**11/08/93 15:59**	**11/09/93 12:59**	**brockett**

Notice that, the assigned and the planned steps for designer badr are rescheduled to both designers brockett and dampier.

```
NOTICE:  The Designer just deleted was busy
         RESCHEDULING his/her tasks.
in-feasible schedule: step # 13
suggested deadline should be >= 20
Would you like to change it?  Answer(y/n)y

Enter the new Deadline 20
in-feasible schedule: step # 12
suggested deadline should be >= 21
Would you like to change it?  Answer(y/n)y

Enter the new Deadline 21
STEP_ID   S_LEVEL  D_NAME      START_TIME  FINISH_TIME
-------   -------  ----------  ----------  -----------
   14     LOW       dampier         3          9
   11     LOW       brockett        8          16
   12     LOW       brockett        16         21
   13     LOW       dampier         16         20
```

D. ANALYSIS OF RESULTS

1. The ECS automatically identifies the affected components by the proposed changes (the primary inputs of the proposed changes) which is very important for software consistency.
2. The ECS creates a substep for each affected module of each approved step as a way to enforce the propagation of the change effects to guarantee software consistency.
3. The ECS gives the manager the edit capability to override the automated decisions which is always needed to give the managers a sense of control over their systems. and also to add the necessary information for the automated function to be performed properly
4. The ECS supports incremental planning and rescheduling of tasks according to the expertise level required by each task, available designers, deadlines constraints, precedence and priority constrains that is needed for medium to large software system that experience large number of changes that involves many designers.
5. The system also has an automated transparent version control and configuration management system that keeps track of the evolution history of the system through tracking the software component versions and which component belongs to which configuration
6. The ECS keeps the information among the three components of its state model consistent all the times to support cooperative work for multi-user, multiple projects organizations.

VI. CONCLUSIONS

A. SUMMARY

In this dissertation, we have presented the Evolution Control System (ECS) as an integrated system for software evolution. We integrate a transparent version control and configuration management mechanism for evolving software systems together with an assignment and scheduling system to enforce cooperation and coordination between designers working concurrently on the same or different systems. This integrated system is necessarily dynamic to cope with the special nature of the software evolution problem where the steps (changes) to be coordinated, scheduled and carried out are only partially known. Time required, the set of sub-tasks for each step, and the input/output constraints between steps are all uncertain, and subject to change as evolution steps are carried out. The ECS introduces the following features:

1. Automated support for changes in plan during the execution of the plan.
2. Automatic decision support for planning and team coordination based on design dependencies captured in the configuration model.
3. Enhanced graph model for software evolution implemented as the main part of the state model of the system.
4. The development and implementation of an automated version control and configuration management mechanism that automatically keeps track of the software component versions and which component belongs to which configuration.
5. The development and implementation of a mechanism for detecting change consequences (determining the components affected by a change) to maintain the global consistency of the design database and provide serializability of updates.
6. The development and implementation of a dynamic heuristic scheduling algorithm that finds a feasible schedule that: meets the deadline and precedence constraints of all the active steps, or suggest new deadlines for the lowest priority deadlines until a feasible schedule that meets the deadlines of the higher priority steps is reached. In addition, the scheduling mechanism supports incremental replanning as additional new steps are created or the attributes of the existing steps are changed.
7. The above features are integrated in such a way that it is possible to use the serialization of tasks according to their precedence constraints as a method for concurrency control as explained in detail in Chapter IV.A.2.

We have found that providing automated support for the following aspects of software evolution is practical and feasible.

1. Changes in the plan during the execution of the plan,
2. Planning and team coordination based on design dependencies,
3. Detecting change consequences,
4. Non-serialized parallel elaborations, and
5. Automated version control and configuration management.

B. IMPORTANCE OF RESEARCH RESULTS

The importance of this research is that it provides managers of the medium to large software projects with the automated help they need to make informed and intelligent decisions in the management of software systems under development/evolution. This is especially important in environments where the number of changes is very large, difficult to follow up manually, hard to coordinate and have their consequences detected and propagated, which in turn threatens the consistency of the software.

Automated help for detecting change consequences and coordinating changes according to the relations and constraints among different changes not only guarantees system consistency, but also guarantees full utilization of the human resources assigned to perform coordinated changes and avoids rollbacks.

Keeping a complete record of the software's evolution history via the developed configuration graph provides a rich history trail for future management reference.

Automatically tracking of software component versions and component configuration dependencies in a way that is transparent to users relieves both managers and designers from the burden of manual book-keeping of the evolution history, which is a very hard, if not impossible, task in medium to large systems.

C. PROPOSED EXTENSIONS

The main proposed extension to this work is to develop and integrate a requirement dependencies mechanism which will propagate changes not only between the specification and implementation components, but also include requirements documents on-line where any change to a software system's requirements automatically triggers proposed changes to corresponding specification modules and subsequently, the implementation modules.

Another plausible refinement is to limit the responsibilities of each designer for a set of projects which constrains the ECS step assignments. Moreover make the scheduler take

into consideration other demands on designers time such as meetings, demos etc. Also include the option of limiting managers responsibilities to a single project.

Other extension is to integrate a change-merging mechanism which will allow the manager to automatically combine results of several completed steps that were explored in parallel. This aspect is currently being explored [22] [23].

Another possible extension is the integration of policies for automatic quality check procedures before the commitment of steps. A minimum step in this direction is to ensure that the software can be compiled without errors before it can be committed to the design database.

As for the implementation of the ECS system the following improvements are required:

1. Integrating the menu driven user interface commands to the existing Tae user interface.
2. Find a way for direct interface between Ada and Ontos DB to save the interface time between Ada to C++ then to Ontos which will significantly enhance the ECS performance.

VII. APPENDICES

A. Formal Specifications

1. State Model and related concepts

```
DEFINITION ECS_state_model
INHERIT configuration_graph
INHERIT designer
INHERIT schedule
STATE ( graph: configuration_graph,
        schedule: schedule_type,
        -- The schedule also is used for optemizing the operations of the
        -- scheduling algorithm.
        primary_input: map {step, set {component_reference}},
        secondary_input: map {step, set {component_reference}},
        affected_modules: map {step, set {component_reference}},
        deadline: map {step,time},
        estimated_duration: map {step, natural},
        precedence: map {step, set {step}},
        priority: map {step, integer},
        status: map {step, step_status},
        step_expertise_level: map {step, exp_level},
        designer_pool: set {designer})
INVARIANT
feasible_schedule (schedule) | manager_notified (schedule),
-- either a feasible schedule is reached or manager is notified of suggestion
-- to get a feasible one.
known_designers (schedule),
-- every designer in the schedule must be in the designer_pool
active_steps (schedule),
```

-- each step in the schedule must be in the configuration graph, must be atomic

-- and its status must be either scheduled or assigned,

-- and the schedule must include all such steps.

single_assignment (schedule)

-- no more than one step is assigned to the same designer at the same time

INITIALLY graph = empty _graph,

designer_pool = { },

schedule = { }

CONCEPT component_reference: type

WHERE Subtype (specific_component_reference, component_reference)

CONCEPT object_id (c: component_reference) VALUE (obj_id: string)

CONCEPT variation_id (c: component_reference) VALUE (var_id: natural)

CONCEPT version_id (c: specific_component_reference) VALUE (v_id: natural)

CONCEPT top_level (c: specific_component_reference) VALUE (b: boolean)

WHERE b <=> ~EXISTS(c1:specific_component_reference :: c part_of c1)

-- prototypes are represented as top_level components

CONCEPT step_status: type

WHERE step_status = enumeration {proposed, approved, scheduled, assigned, abandoned, completed, all}

-- all is used to indicate all the steps of a certain prototype disregarding their status

CONCEPT exp_level:: type

WHERE exp_level = enumeration {low, medium, high, none}

-- none is used as a default value for the expertise_level of a step and treated as

-- low to distinguish between the values entered by the manager and the default

-- values.

```
CONCEPT feasible_schedule (s: schedule_type) VALUE (b: boolean)
    WHERE b <=> known_designers (s) & single assignment (s) & active_steps (s)
    & predecessors_finished (s)& on_time (s, deadline) & sufficient_expertise (s)
CONCEPT known_designers (s: schedule_type) VALUE (b: boolean)
    WHERE b <=> ALL (d: designer SUCH THAT d IN s:: d IN designer_pool)
    -- every designer in the schedule must be in the designer_pool

CONCEPT single_assignment (schedule: schedule_type) VALUE (b: boolean)
    WHERE b <=> ALL (s1, s2: step:: s1 IN schedule & s2 IN schedule &
    schedule (s1).designer = schedule (s2).designer &
    schedule (s1).scheduled_finish_time > schedule(s2).scheduled_start_time
    > schedule(s1).scheduled_start_time => s1= s2)
    -- no more than one step is assigned to the same designer at the same time

CONCEPT active_steps (s: schedule_type) VALUE (b: boolean)
    WHERE b <=> ALL (st: step:: st IN s <=> step_in_graph (st, graph) & atomic (st)
    & status (st) IN {scheduled, assigned})
    -- each step in the schedule must be in the configuration graph, must be atomic,
    -- and its status either scheduled or assigned, and the schedule must include all
    -- such steps

CONCEPT atomic (st: step) VALUE (b: boolean)
    WHERE b <=> ~ EXISTS (st1: step :: st1 part_of st)
    -- A step is atomic if it has no substeps.

CONCEPT predecessors_finished (schedule: schedule_type)
    VALUE (b: boolean)
    WHERE b <=> ALL (st1, st2: step SUCH THAT st1 IN schedule &
```

```
        st2 IN schedule & precedes (st1, st2, dep_graph) ::
        schedule(st1).scheduled_finish_time <= schedule(st2).scheduled_start_time)

    CONCEPT on_time (schedule: schedule_type, d: map {step,time})
        VALUE (b: boolean)
        WHERE b <=> ALL (st: step SUCH THAT st IN schedule:: schedule
        (st).scheduled_finish_time <= d (st))

    CONCEPT sufficient_expertise (schedule: schedule_type) VALUE (b: boolean)
        WHERE b <=> ALL (st: step SUCH THAT st IN schedule::
        step_expertise_level (st) <= designer_expertise_level (schedule (st).designer))
        -- a designer can be assigned to a step only if his expertise_level is at least
        -- that of the assigned step

    CONCEPT unique_step_id (s: step) VALUE (b: boolean)
        WHERE b <=> ALL (s1: step:: step_id (s1) = step_id (s) => s1= s)
        -- step_id is unique for all steps.

END
```

a. Configuration graph

```
DEFINITION configuration_graph
    INHERIT component_node
    INHERIT step_node
    MODEL (component_nodes: set {version}, step_nodes: set {step},
            part_of_edges, used_by_edges, I_edges, O_edges: set {pair})
     INVARIANT
        ALL (g: configuration_graph, e: pair SUCH THAT e IN g.part_of_edges::
        [(e.start IN g.component_nodes & e.end IN g.component_nodes) |
        (e.start IN g.step_nodes & e.end IN g.step_nodes)]
```

```
        | e IN g.used_by_edges:: e.start IN g.component_nodes & e.end IN
        g.component_nodes | e IN g.I_edges:: e.start IN g.component_nodes & e.end IN
        g.step_nodes
        | e IN g.O_edges:: e.start IN g.step_nodes & e.end IN g.component_nodes)

CONCEPT is_component_node (x: version, g: configuration_graph)
VALUE (b: boolean)
WHERE b <=> x IN g.component_nodes

CONCEPT component_in_graph (o: component_reference, g: configuration_graph)
        VALUE (b: boolean)
          WHERE b <=> SOME (c: version SUCH THAT is_component_node (c, g)::
          object_id (c) = object_id(o))

CONCEPT step_in_graph (s: step, g: configuration_graph) VALUE (b: boolean)
          WHERE b <=> s IN g.step_nodes

CONCEPT empty_graph VALUE (g: configuration_graph)
          WHERE g.nodes = { }, g.edges = { }

CONCEPT add_step_node (s: step, g1: configuration_graph)
-- Add a step node.
VALUE (g2: configuration_graph)
WHERE g2.step_nodes = g1.step_nodes U {s}, g2.component_nodes =
            g1.component_nodes, g2.edges = g1.edges

CONCEPT add_component_node (c: version, g1: configuration_graph)
-- Add a component node.
```

VALUE (g2: configuration_graph)

WHERE g2.component_nodes = g1.component_nodes U {c}, g2.step_nodes =

g1.step_nodes, g2.edges = g1.edges

CONCEPT add_input_edge (c: version, s: step, g1: configuration_graph)

-- Add an input edge.

VALUE (g2: configuration_graph)

WHERE g2.component_nodes = g1.component_nodes U {c}, g2.step_nodes =

g1.step_nodes U {s}

g2.I_edges = g1.I_edges U {[start:: c, end:: s]}, g2.O_edges = g1.O_edges,

g2.part_of_edges = g1.part_of_edges, g2.used_by_edges = g1.used_by_edges

CONCEPT add_output_edge (s: step, c: version, g1: configuration_graph)

-- Add an input edge.

VALUE (g2: configuration_graph)

WHERE g2.component_nodes = g1.component_nodes U {c},

g2.step_nodes = g1.step_nodes U {s}

g2.I_edges = g1.I_edges, g2.O_edges = g1.O_edges U {[start:: s, end:: c]},

g2.part_of_edges = g1.part_of_edges, g2.used_by_edges = g1.used_by_edges

CONCEPT remove_input_edge (c: component, s: step, g1: configuration_graph)

-- Remove an input edge.

VALUE (g2: configuration_graph)

WHERE g2.nodes = g1.nodes,

g2.I_edges = g1.I_edges - {[start:: c, end:: s]}, g2.O_edges = g1.O_edges,

g2.part_of_edges = g1.part_of_edges, g2.used_by_edges = g1.used_by_edges

CONCEPT remove_input_edges (s, g1) VALUE (g2: configuration_graph), WHERE

g2.nodes = g1.nodes,

```
    g2.I_edges = g1.I_edges - {ALL (e: I_edge SUCH THAT end (e)= s)},
    g2.O_edges = g1.O_edges, g2.part_of_edges = g1.part_of_edges,
    g2.used_by_edges = g1.used_by_edges

    CONCEPT configuration_graph_node: type
    WHERE Subtype (version, configuration_graph_node),
          Subtype (step, configuration_graph_node),
          ALL (n: configuration_graph_node :: n IN version | n IN step)
          -- configuration graph nodes are either steps or versions.
    CONCEPT pair: type
     WHERE pair = tuple {start, end:: configuration_graph_node }
    CONCEPT version: specific_component_reference
END
```

```
DEFINITION step_node          -- Concepts for describing step_node
    CONCEPT step: type
    CONCEPT step_id (s: step) VALUE (s_id: natural)
    CONCEPT top_level(s: step) VALUE (b: boolean)
    WHERE b <=> ~EXISTS(s1: step:: s part_of s1)
END
```

```
DEFINITION dependency_graph          -- Concepts for describing
dependency_graph
    dep_graph= f(dep_nodes, dep_edges)
    CHOOSE (dep_nodes, dep_edges SUCH THAT
          ALL(n:: n  ∈ dep_nodes <=> n IN graph.step_nodes &
               status(step(n)) IN {scheduled, assigned} & atomic (step(n))) &
          ALL(e:: e  ∈  dep_edges <=> e.start, e.end IN dep_nodes ))
END
```

---

b. Designer_Pool

DEFINITION designer -- Concepts for describing designer_pool

CONCEPT designer: type

CONCEPT name (d: designer) VALUE (n: string)

CONCEPT designer_expertise_level (d: designer) VALUE (e: exp_level)

CONCEPT status (d: designer) VALUE (s: enumeration {busy, free})

END

---

c. Schedule

DEFINITION schedule -- Concepts for describing the schedule

CONCEPT schedule_type: type

WHERE schedule_type = map (s: step, tuple {d:: designer, scheduled_start_time, scheduled_finish_time: time})

CONCEPT update_schedule (old_schedule: schedule_type, s: step SUCH THAT status (s) = approved)

VALUE (new_schedule: schedule_type)

-- recalculate schedule due to scheduling a new step

WHERE ALL (s1: step SUCH THAT s1 part_of s & atomic (s1)::

s1 IN new_schedule) & ALL (s1: step:: s1 IN old_schedule => s1 IN new_schedule)

CONCEPT update_schedule (old_schedule: schedule_type, s: step SUCH THAT status (s) IN {abandoned, completed})

VALUE (new_schedule: schedule_type)

-- recalculate schedule due to abandoning a step

WHERE ALL (s1: step SUCH THAT s1 part_of s & atomic (s1)::

~ (s1 IN new_schedule) & EXISTS (s2 IN new_schedule SUCH THAT

new_schedule (s2).designer = old_schedule (s1).designer & status (s2) = assigned)) &

ALL (s2: step:: s2 IN old_schedule <=> s2 IN new_schedule | s2 part_of s)

```
CONCEPT update_schedule (old_schedule: schedule_type, d: designer SUCH THAT
~ (d IN old_schedule))
VALUE (new_schedule: schedule_type)
-- recalculate schedule after adding a new designer
WHERE (d IN new_schedule & ALL (s1: step:: s1 IN new_schedule <=> s1 IN
old_schedule))

CONCEPT update_schedule (old_schedule: schedule_type, d: designer SUCH THAT
                                    d IN old_schedule)
VALUE (new_schedule: schedule_type)
-- recalculate schedule after dropping a designer
WHERE (~(d IN new_schedule) & ALL (s1: step:: s1 IN new_schedule <=>
                s1 IN old_schedule))

CONCEPT schedule_changes (old_schedule, new_schedule: schedule_type)
VALUE (ch: schedule_type)
WHERE ALL (s1: step:: s1 IN ch <=> (s1 IN old_schedule & ~ (s1 IN
new_schedule)) | (~ (s1 IN old_schedule) & s1 IN new_schedule) |
(old_schedule [s1] ~= new_schedule [s1] & ch (s1) = new_schedule [s1]))
END
```

---

d. Assignments

```
DEFINITION assignment                    -- concepts for assigning a designer to
a step
INHERIT ECS_state_model
INHERIT designer
CONCEPT curr_assign (s: step , d: designer) VALUE (b: boolean)
```

WHERE b <=> active (s) & predecessor_finished (s) & status (d) = free & designer_expertise_level (d) >= step_expertise_level (s)

CONCEPT predecessor_finished (s: step) VALUE (b: boolean)

WHERE b <=> ALL (s1: step SUCH THAT active (s1) ::

~precedes(s1, s, dep_graph))

CONCEPT active (s: step) VALUE (b: boolean)

WHERE b <=> status (s) IN {scheduled, assigned}

CONCEPT check_due_completion (s: schedule_type) VALUE (w: warning, ss: set {step})

WHERE ALL (st: step :: st IN ss <=> st IN s & status (st) = assigned & s(st).scheduled_finish_time <= current_time)

CONCEPT warning: string

WHERE warning = "the following steps missed their finish time, either commit or increase their estimated duration"

END

---

2. Behavior Model

MACHINE Evolution_Control_System

INHERIT designer_interface

INHERIT manager_interface

END

MACHINE common_interface

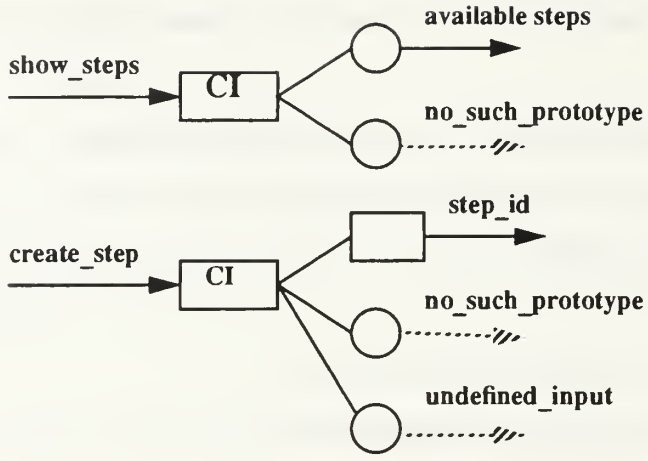


FIGURE 31. Stimulus_Response diagram for the common interface

INHERIT ECS_state_model

MESSAGE show_steps (p: component_reference SUCH THAT top_level(p),
t: step_status)
-- display all steps with the given status of the given prototype
WHEN component_in_graph(p, graph)
REPLY (s: sequence {step})
WHERE ALL (i: step:: i IN s <=> step_in_graph(i, graph) & (status (i) = t | t=all))
OTHERWISE REPLY EXCEPTION no_such_prototype

MESSAGE create_step (p: component_reference SUCH THAT top_level(p),
primary_input : set {component_reference})
WHEN component_in_graph(p, graph)&
component_in_graph (c: component_reference SUCH THAT c IN primary_input,
graph)
REPLY (s: step

WHERE unique_step_id (s)

TRANSITION primary_input =bind(s, prim, *primary_input),

secondary_input = bind(s, sec,*secondary_input),

affected_modules = bind (s, aff, *affected_modules),

-- affected modules and secondary inputs are automatically generated

add_step_node(s, graph), only_change (graph, *graph, s)

WHEN component_in_graph(p, graph)&

~ component_in_graph (c: component_reference SUCH THAT c IN primary_input, graph)

REPLY EXCEPTION undefined_input

OTHERWISE REPLY EXCEPTION no_such_prototype

END

MACHINE edit_interface

-- This interface includes all the messages required to modify the step attributes, the

-- management constraints, and the designer_pool data.

INHERIT ECS_state_model

MESSAGE add_primary_input (i: step SUCH THAT top_level (i),

c: specific_component_reference)

-- this is used to add primary_input to the step as they become known

-- normally a step has one primary input except in case of a merge when it has

-- two or more

WHEN ~ component_in_graph (c, graph)

REPLY EXCEPTION undefined_input

WHEN step_in_graph(i,graph) & component_in_graph (c, graph)

REPLY done

TRANSITION

primary_input = bind(i, *primary_input (i) U {c}, *primary_input),

```
        only_change (i, *i, c)
OTHERWISE REPLY EXCEPTION no_such_step

---

MESSAGE add_affected_modules (i: step SUCH THAT top_level (i),
                              c: component_reference)
    -- this is used to add affected_modules to the step as they become known
    -- since the calculated set is an approximation
    WHEN ~ component_in_graph (c, graph)
    REPLY EXCEPTION undefined_input
    WHEN step_in_graph(i,graph) & ~ (i IN schedule)
    REPLY done
    TRANSITION only_change (i, *i, c),
    affected_modules = bind(i, *affected_modules (i) U {c}, *affected_modules)
    WHEN i IN schedule & feasible_schedule (update_schedule (*schedule, i))
    REPLY (schedule_changes (*schedule, schedule))
    -- the schedule must be updated to maintain the invariant
    TRANSITION only_change (i, *i, c), only_change(*schedule, schedule, i),
    affected_modules = bind(i, *affected_modules (i) U {c}, *affected_modules)
    WHEN i IN schedule & SOME (s: schedule_type SUCH THAT
    s = update_schedule (*schedule, i):: ~ feasible_schedule (s) &
    ~ manager_confirmation (s))
    -- ask for manager confirmation only if a feasible schedule cannot be reached
    REPLY changes_undone
    WHEN i IN schedule & SOME (s: schedule_type SUCH THAT
        s = update_schedule (*schedule, i)::
        ~ feasible_schedule (s) & manager_confirmation (s))
        REPLY (schedule_changes (*schedule, schedule))
        TRANSITION only_change (i, *i, c), only_change(*schedule, schedule, i),
```

affected_modules = bind(i, *affected_modules (i) U {c}, *affected_modules)

deadline = least_slips (*deadline, schedule)

-- the schedule must be updated to maintain the invariant

-- The deadline changes is kept to the minimum required to get a feasible

-- schedule

OTHERWISE REPLY EXCEPTION no_such_step

MESSAGE add_secondary_input (i: step SUCH THAT top_level (i),

c: component_reference)

-- this is used to add secondary_input to the step as they become known

WHEN ~ component_in_graph (c, graph)

REPLY EXCEPTION undefined_input

WHEN step_in_graph(i,graph) & ~ (i IN schedule)

REPLY done

TRANSITION only_change (i, *i, c),

secondary_input = bind(i, *secondary_input (i) U {c}, *secondary_input)

WHEN i IN schedule & feasible_schedule (update_schedule (*schedule, i))

REPLY (schedule_changes (*schedule, schedule))

-- the schedule must be updated to maintain the invariant

TRANSITION only_change (i, *i, c), only_change(*schedule, schedule, i),

secondary_input = bind(i, *secondary_input (i) U {c}, *secondary_input)

WHEN i IN schedule & SOME (s: schedule_type SUCH THAT

s = update_schedule (*schedule, i):: ~ feasible_schedule (s) &

~ manager_confirmation (s))

-- ask for manager confirmation only if a feasible schedule cannot be reached

REPLY changes_undone

WHEN i IN schedule & SOME (s: schedule_type SUCH THAT

s = update_schedule (*schedule, i)::

~ feasible_schedule (s) & manager_confirmation (s))

REPLY (schedule_changes (*schedule, schedule))

TRANSITION only_change (i, *i, c), only_change(*schedule, schedule, i),

secondary_input = bind(i, *secondary_input (i) U {c}, *secondary_input) ,

deadline = least_slips (*deadline, schedule)

-- the schedule must be updated to maintain the invariant

-- The deadline changes is kept to the minimum required to get a feasible

-- schedule

OTHERWISE REPLY EXCEPTION no_such_step

MESSAGE delete_primary_input (i: step SUCH THAT top_level (i),

c: component_reference)

-- this is used to delete inputs from the step's primary input.

WHEN WHEN ~ component_in_graph (c, graph)

REPLY undefined_input

WHEN step_in_graph(i,graph)

REPLY done

TRANSITION primary_input = bind(i, *primary_input (i) - {c},

*primary_input),

only_change (i, *i, primary_input)

OTHERWISE REPLY EXCEPTION no_such_step

MESSAGE delete_affected_modules (i: step SUCH THAT top_level (i),

c: component_reference)

-- this is used to delete those affected_modules that do not need change.

WHEN ~ component_in_graph (c, graph)

REPLY undefined_object

WHEN step_in_graph(i,graph) & status (i) = approved

```
        REPLY done
        TRANSITION only_change (i, *i, c),
        affected_modules = bind(i, *affected_modules (i)- {c}, *affected_modules)
    WHEN i IN schedule
        REPLY (schedule_changes (*schedule, schedule))
        TRANSITION  only_change (i, *i, cs),
        affected_modules = bind(i, *affected_modules (i)- {c}, *affected_modules)
    OTHERWISE REPLY EXCEPTION no_such_step
```

```
MESSAGE delete_secondary_input (i: step SUCH THAT top_level (i),
                                c: component_reference)
    -- this is used to delete secondary inputs to a step
    WHEN ~ component_in_graph (c, graph)
        REPLY EXCEPTION undefined_input
    WHEN step_in_graph(i,graph) & ~ (i IN schedule)
        REPLY done
        TRANSITION only_change (i, *i, c),
        secondary_input = bind(i, *secondary_input (i) - {c}, *secondary_input)
    WHEN i IN schedule
        REPLY (schedule_changes (*schedule, schedule))
        TRANSITION only_change (i, *i, secondary_input),
        secondary_input = bind(i, *secondary_input (i) - {c}, *secondary_input)
        -- the schedule must be updated to maintain the invariant
    OTHERWISE REPLY EXCEPTION no_such_step
```

```
MESSAGE update_priority (i: step SUCH THAT top_level (i), p: natural)
    WHEN i IN steps & (p > priority (i1) SUCH THAT i1 IN precedence (i))
        REPLY EXCEPTION priority_conflict
    WHEN step_in_graph(i,graph) & ~ (i IN schedule)
```

```
        REPLY done
        TRANSITION priority = bind (i, p, *priority), only_change (i, *i, p)
    WHEN i IN schedule & feasible_schedule(update_schedule (*schedule, i))
        REPLY (schedule_changes (*schedule, schedule))
        TRANSITION priority = bind (i, p, *priority)
        -- the schedule must be updated to maintain the invariant
    WHEN i IN schedule & SOME (s: schedule_type SUCH THAT
        s = update_schedule (*schedule, i):: ~ feasible_schedule (s) &
        ~ manager_confirmation (s))
        REPLY change_undone
    WHEN i IN schedule & SOME (s: schedule_type SUCH THAT
        s = update_schedule (*schedule, i):: ~ feasible_schedule (s) &
         manager_confirmation (s))
        REPLY (schedule_changes (*schedule, schedule))
        TRANSITION priority = bind (i, p, *priority),
        deadline = least_slips (*deadline, schedule)
        -- the schedule must be updated to maintain the invariant
        -- The deadline changes is kept to the minimum required to get a feasible
        -- schedule
    OTHERWISE REPLY EXCEPTION no_such_step

-------------------------------------------------------------------

MESSAGE update_precedence (i: step SUCH THAT top_level (i), p: set {step})
    WHEN step_in_graph(i,graph) & (priority (i) > priority (i1) SUCH THAT
    i1 IN p)
        REPLY EXCEPTION priority_conflict
    WHEN step_in_graph(i,graph)& ~ (i IN schedule)
        REPLY done
        TRANSITION precedence = bind (i, p, *precedence), only_change (i, *i, p)
    WHEN i IN schedule & feasible_schedule (update_schedule (*schedule, i))
```

```
        REPLY (schedule_changes (*schedule, schedule))
        -- the schedule must be updated to maintain the invariant
        TRANSITION precedence = bind (i, p, *precedence)
    WHEN i IN schedule & SOME (s: schedule_type SUCH THAT
        s = update_schedule (*schedule, i):: ~ feasible_schedule (s) &
        ~ manager_confirmation (s))
        -- ask for manager confirmation only if schedule invalidated
        REPLY change_undone
    WHEN i IN schedule & SOME (s: schedule_type SUCH THAT
        s = update_schedule (*schedule, i):: ~ feasible_schedule (s)&
         manager_confirmation (s))
        REPLY (schedule_changes (*schedule, schedule))
        TRANSITION precedence = bind (i, p, *precedence),
            deadline = least_slips (*deadline, schedule)
        -- the schedule must be updated to maintain the invariant
        -- The deadline changes is kept to the minimum required to get a feasible
                    schedule
    OTHERWISE REPLY EXCEPTION no_such_step
```

```
MESSAGE update_deadline (i: step SUCH THAT top_level (i), d:time)
    WHEN step_in_graph(i,graph) &~ (i IN schedule)
        REPLY done
        TRANSITION deadline = bind (i, d, deadline)
    WHEN i IN schedule & feasible_schedule (update_schedule (*schedule, i))
        REPLY (schedule_changes (*schedule, schedule))
        TRANSITION deadline = bind (i, d, deadline)
        -- the schedule must be updated to maintain the invariant
    WHEN i IN schedule & SOME (s: schedule_type SUCH THAT
        s = update_schedule (*schedule, i):: ~ feasible_schedule (s) &
        ~ manager_confirmation (s))
```

REPLY change_undone

WHEN i IN schedule & SOME (s: schedule_type SUCH THAT

s = update_schedule (*schedule, i):: ~ feasible_schedule (s) &

manager_confirmation (s))

REPLY (schedule_changes (*schedule, schedule))

TRANSITION deadline = least_slips (*deadline, schedule)

-- the schedule must be updated to maintain the invariant

-- The deadline changes is kept to the minimum required to get a feasible

-- schedule

OTHERWISE REPLY EXCEPTION no_such_step

MESSAGE update_estimated_duration (i: step, t: natural)

WHEN step_in_graph(i,graph) & ~ (i IN schedule)

REPLY done

TRANSITION estimated_duration = bind (i, t, *estimated_duration)

WHEN i IN schedule & feasible_schedule(update_schedule (*schedule, i))

REPLY (schedule_changes (*schedule, schedule))

TRANSITION estimated_duration = bind (i, t, *estimated_duration)

-- the schedule must be updated to maintain the invariant

WHEN i IN schedule & SOME (s: schedule_type SUCH THAT

s = update_schedule (*schedule, i):: ~ feasible_schedule (s) &

~ manager_confirmation (s))

REPLY change_undone

WHEN i IN schedule & SOME (s: schedule_type SUCH THAT

s = update_schedule (*schedule, i):: ~ feasible_schedule (s) &

manager_confirmation (s))

REPLY (schedule_changes (*schedule, schedule))

TRANSITION estimated_duration = bind (i, t, *estimated_duration),

deadline = least_slips (*deadline, schedule)

-- the schedule must be updated to maintain the invariant

```
        -- The deadline changes is kept to the minimum required to get a feasible
        -- schedule
    OTHERWISE REPLY EXCEPTION no_such_step

CONCEPT consistent_deadlines (d1, d2: map {step,time}) VALUE (b: boolean)
    WHERE b <=> ALL (s: step:: d1(s) = d2 (s))
    -- for d1 to be consistent with d2, either the deadline of each step in d1 is
    -- equal to that of the same step in d2 or relaxed to be equal to its calculated
    -- finish time.

CONCEPT valid_slip (d1, d2: map {step,time}) VALUE (b: boolean)
    WHERE b <=> consistent_deadlines (d1, d2) &
    ALL (s1, s2: step::
    d1(s1) = none ~= d2(s1) & d1(s2) ~= none => priority (s2) >= priority (s1))
    -- new deadline map is valid only if the relaxed deadlines are the lowest
    -- priority deadlines

CONCEPT slips_more (d1, d2: map {step,time}) VALUE (b: boolean)
    WHERE b <=> consistent_deadlines (d1, d2) & d1 ~= d2

CONCEPT least_slips (d1: map {step,time}, s: schedule_type)
    VALUE (d2: map {step,time})
    WHERE valid_slip (d2,d1) & feasible_schedule (s, d2) &
    ALL (d3: map {step,time}::
    slips_more (d2, d3) & consistent_deadlines (d3,d1) =>
    ~ SOME (s1: schedule_type::feasible_schedule (s1, d3))

CONCEPT only_change (x y: any, $attribute_names: identifier)
 VALUE (b: boolean)
```

```
    -- True if x and y can differ only in the listed attributes.
    WHERE ALL (id: identifier:: x.id ~= y.id => id IN attribute_names)

END
```

---

3. Manager Interface

```
MACHINE designer_pool_view
    INHERIT ECS_state_model
    INHERIT common_interface
    MESSAGE add_designer (d: designer)
    -- adding a designer to the designer_pool
        WHEN ~ (d IN designers_pool)
            REPLY (schedule_changes (*schedule, schedule))
            TRANSITION designers_pool = *designers_pool U {d}
            -- the schedule must be updated to use the new designer and maintain the
            -- invariant
            -- adding a designer cannot invalidate a feasible schedule
        OTHERWISE REPLY EXCEPTION designer_exists
```

```
    MESSAGE drop_designer (d: designer)
        -- remove designer from the designers_pool
        WHEN d IN designer_pool &~ (d IN schedule)
            REPLY done
            TRANSITION
            designers_pool = *designers_pool - {d}
        WHEN d IN designer_pool & d IN schedule
            REPLY warning_confirmation_required
        WHEN d IN designer_pool & d IN schedule &
        SOME (s: schedule_type SUCH THAT s= update_schedule (*schedule, d)::
```

feasible_schedule (s) & manager_confirmation (s))

REPLY (schedule_changes (*schedule, schedule))

TRANSITION designer_pool = *designer_pool - {d}

WHEN d IN designer_pool & d IN schedule & SOME (s: schedule_type SUCH THAT

s = update_schedule (*schedule, d):: ~ feasible_schedule (s) &

~ manager_confirmation (s))

REPLY change_undone

WHEN d IN designer_pool & d IN schedule &

SOME (s: schedule_type SUCH THAT s = update_schedule (*schedule, d)::

~ feasible_schedule (s) & manager_confirmation (s))

REPLY (schedule_changes (*schedule, schedule))

TRANSITION designers_pool = *designers_pool - {d},

deadline = least_slips (*deadline, schedule)

-- the schedule must be updated to maintain the invariant

-- The deadline changes is kept to the minimum required to get a feasible

-- schedule

OTHERWISE REPLY EXCEPTION no_such_designer

MESSAGE designer_expertise_level (d: designer, e: exp_level)

-- modify the designer's expertise level

WHEN d IN designer_pool & ~ (d IN schedule)

REPLY done

TRANSITION designer_expertise_level (d) = e,

only_change (*d, d, designer_expertise_level)

WHEN d IN designer_pool & d IN schedule &

feasible_schedule (update_schedule (*schedule, d))

REPLY (schedule_changes (*schedule, schedule))

TRANSITION designer_expertise_level (d) = e,

```
            only_change (*d, d, designer_expertise_level)
        WHEN d IN designer_pool & d IN schedule &
         SOME (s: schedule_type SUCH THAT s= update_schedule (*schedule, d)::
            ~ feasible_schedule (s) & ~ manager_confirmation (s))
            REPLY change_undone
        WHEN d IN designer_pool & d IN schedule &
         ALL (s: schedule_type SUCH THAT s= update_schedule (*schedule, d)::
            ~ feasible_schedule (s) & manager_confirmation (s))
            REPLY  (schedule_changes (*schedule, schedule))
            TRANSITION designer_expertise_level (d) = e,
            only_change (*d, d, designer_expertise_level)
            deadline = least_slips (*deadline, schedule)
            -- the schedule must be updated to maintain the invariant
            -- The deadline changes is kept to the minimum required to get a feasible
            -- schedule
    OTHERWISE REPLY EXCEPTION no_such_designer

    MESSAGE show_designers (dp: set{designer})
    -- display all the designers in the designer's pool
        WHEN dp /= { }
            REPLY (s: set {designers})
            WHERE ALL (d: designer :: d IN s <=> d IN dp)
    OTHERWISE REPLY EXCEPTION no_such_prototype

END
```

MACHINE manager_interface

```
    INHERIT user -- defines User_class, uses
    INHERIT designer_pool_view
```

INHERIT common_interface

INHERIT edit_interface

INHERIT ECS_state_model

MESSAGE approve_step (s: step SUCH THAT top_level(s))

-- used to approve a step which triggers the change propagation via the

-- calculation of the affected modules

WHEN status (s) = proposed

REPLY (ss: set (component_reference))

TRANSITION status = bind (s, approved, *status), auto_create_substep (s)

-- advance the step status to "approved"

-- calculate the set of modules affected by the step

-- create an atomic substep for each affected module

OTHERWISE REPLY EXCEPTION no_such_proposed_step

MESSAGE show_schedule (p: prototype_name)

-- display the full schedule

WHEN prototype (p) IN prototypes

REPLY (s: schedule_type)

WHERE s = schedule

OTHERWISE REPLY EXCEPTION no_such_prototype

MESSAGE schedule_step (s: step)

WHEN status (s) = approved &

SOME (s_1: step SUCH THAT s_1 part_of s:: estimated_duration (s_1) = 0)

REPLY estimated_duration_not_specified

WHEN status (s) = approved & feasible_schedule (update_schedule (*schedule, s))

```
        REPLY (schedule_changes (*schedule, schedule))
        TRANSITION status = bind (s, scheduled, *status)
        -- the schedule must be updated to maintain the invariant
    WHEN status (s) = approved & SOME (sch: schedule_type SUCH THAT
        sch = update_schedule (*schedule, s)::~ feasible_schedule (sch) &
        ~ manager_confirmation (sch))
        REPLY step_is_not_scheduled
    WHEN status (s) = approved & SOME (sch: schedule_type SUCH THAT
        sch = update_schedule (*schedule, s):: ~ feasible_schedule (sch) &
         manager_confirmation (sch))
        REPLY (schedule_changes (*schedule, schedule))
        TRANSITION status = bind (s, scheduled, *status)
        deadline = least_slips (*deadline, schedule)
         -- the schedule must be updated to maintain the invariant
        -- The deadline changes are kept to the minimum required to get a feasible
        -- schedule
OTHERWISE REPLY EXCEPTION no_such_approved_step
```

```
MESSAGE abandon_step (s: step)
    WHEN step_in_graph(s,graph) & ~ (s IN schedule)
        REPLY done
        TRANSITION status = bind (s, abandoned, *status)
    WHEN s IN schedule & status (s) = scheduled
        REPLY (schedule_changes (*schedule, schedule))
        TRANSITION status = bind (s, abandoned, *status)
         -- the schedule must be updated to maintain the invariant
    WHEN s IN schedule & status (s) = assigned
        SEND ("step abandoned", s) TO schedule [s].designer
        REPLY (schedule_changes (*schedule, schedule))
```

```
        TRANSITION status = bind (s, abandoned, *status),
        remove_input_edges (s, graph)
        -- remove the abandoned step from the schedule and re-assign its designer
        -- remove version bindings of inputs to the step
    OTHERWISE REPLY EXCEPTION no_such_step
```

```
MESSAGE suspend_step (s: step)
    WHEN s IN schedule & status (s) = scheduled
        REPLY (schedule_changes (*schedule, schedule))
        TRANSITION status = bind (s, approved, *status)
        -- the schedule must be updated to maintain the invariant
    WHEN s IN schedule & status (s) = assigned
        SEND ("step suspended", s) TO schedule (s).designer
        REPLY (schedule_changes (*schedule, schedule))
        TRANSITION status = bind (s, approved, *status),
                        remove_input_edges (s, graph)
        -- the schedule must be updated to maintain the invariant
OTHERWISE REPLY EXCEPTION no_such_scheduled_step
```

```
MESSAGE commit_step (s: step SUCH THAT top_level(s))
-- committing a top level step means adding its output components to the configuration graph
-- and releasing the committed version to the public use.
    WHEN status (s) = assigned &
        ALL (s1: step SUCH THAT part_of (s1, s):: status (s1) = completed)
        REPLY done
        TRANSITION status (s) = completed,
        -- The new protoytpe configuration should be created at this point
```

```
OTHERWISE REPLY EXCEPTION no_such_assigned_step

---

 MESSAGE manager_confirmation (s: schedule_type)
REPLY (b: boolean)
-- b is true if the manager responds positively after being notified of
-- schedule invalidation (deadline_change).

---

CONCEPT manager_notified (s: schedule_type)
VALUE (dc: set {deadline_change })
WHERE ALL(st: step:: st IN dc <=> deadline (st) < estimated_finish_time(st))

CONCEPT deadline_change: type
WHERE deadline_change =
                    tuple {st:: step, deadline (st) estimated_finish_time (st):: time}

CONCEPT auto_create_substep (s: step) VALUE (ss: set {step})
WHERE ALL (st: step, c: component_reference :: st IN ss <=> c IN affected_modules
        (s) & primary_input(st) = c & atomic (st))

CONCEPT manager: User_class
WHERE ALL (m: manager:: uses (m, ECS))
-- managers use the ECS
manages (m, designers)
manages (m, steps)
END

---
```

4. Designer Interface

The designer interface with ECS enables the designer to view the steps in a given prototype with a given status and get the sub-steps assigned to him. This interface also enables the designer to create a step or a sub-step of an assigned step, updating any of the step attributes, as well as committing the assigned sub-step.

MACHINE designer_interface

INHERIT common_interface

INHERIT ECS_state_model

MESSAGE commit_step (s: step, cc : set {specific_component_reference})

-- committing an atomic step means storing its output component in the

-- shared data space, configuring it, and making it visible only to the design team

WHEN status (s) = assigned & current_time <= scheduled_finish_time &

ALL (o: object:: o IN cc & designer_confirmation (o) & complete (o))

REPLY done

TRANSITION ALL (o: object:: o IN cc:: add_output_edge (s, o, graph),

status = bind (s, completed, *status)

WHEN status (s) = assigned & current_time <= scheduled_finish_time &

ALL (o: object:: o IN cc & designer_confirmation (o) & ~ complete (o))

REPLY (ss: sequence {step})

WHERE ALL (o: object SUCH THAT o IN cc & ~complete (o)::

EXIST (s1: step SUCH THAT s1 IN ss & primary_input (s1) = o &

status (s1) = proposed))

TRANSITION ALL (o: object:: o IN cc:: add_output_edge (s, o, graph),

status = bind (s, completed, *status)

WHEN status (s) = assigned & current_time > scheduled_finish_time &

ALL (o: object:: o IN cc & designer_confirmation (o) & complete (o))

REPLY (schedule_changes (*schedule, schedule))

TRANSITION ALL (o: object:: o IN cc:: add_output_edge (s, o, graph),

```
        status = bind (s, completed, *status)
        -- schedule must be updated to maintain the invariant
    WHEN status (s) = assigned & current_time > scheduled_finish_time &
        ALL (o: object:: o IN cc & designer_confirmation (o) & ~complete (o))
        REPLY (schedule_changes (*schedule, schedule), ss: sequence {step})
        WHERE ALL (o: object SUCH THAT o IN cc & ~complete (o) ::
        EXIST (s1: step SUCH THAT s1 IN ss & primary_input (s1) = o &
        status (s1) = proposed))
        TRANSITION ALL (o: object:: o IN cc:: add_output_edge (s, o, graph),
        status = bind (s, completed, *status)
        -- schedule must be updated to maintain the invariant
OTHERWISE REPLY EXCEPTION no_such_assigned_sub-step

---

MESSAGE designer_confirmation (s: schedule_type)
    REPLY (b: boolean)
        -- b is true if the designer responds positively after a warning of schedule
        -- invalidation

---

CONCEPT designer: User_class
    WHERE ALL (d: designer:: uses (d, ECS))
    --designers use the ECS
    ALL (s: step:: SOME (d: designer:: perform (d, s)))
END
```

---

5. Type Time

```
    INHERIT equality{date}
    MODEL (day, month, year, hour, minute: nat)
    INVARIANT ALL (d:time:: 1 <= d.day <= 31 & 1 <= d.month <= 12 &
```

```
0 <= d.year <= 99 & 0 <= d.hour <= 23 & 0 <= d.minute <= 59 &d < none)
MESSAGE create (d m y, h, min: nat)
        WHEN 1 <= d <= 31 & 1 <= m <= 12 & 0 <= y <= 99 &
                0 <= h <= 23 & 0 <= min <= 59
        REPLY (d1:time)
        WHERE d1.day = d, d1.month = m, d1.year = y, d1.hour =h,
                d1.minute = min
        OTHERWISE REPLY EXCEPTION illegal_date

MESSAGE equal (d1 d2:time)
        REPLY (b: boolean)
        WHERE b <=> d1.day = d2.day & d1.month = d2.month &
                d1.year = d2.year & d1.hour = d2.hour &
                d1.minute = d2.minute
MESSAGE "<"(d1 d2:time)
        REPLY (b: boolean)
        WHERE b <=> 0 < (d2.year - d1.year) MOD 100 < 50
        | d1.year = d2.year & d1.month < d2.month
        | d1.year = d2.year & d1.month = d2.month & d1.day < d2.day
        | d1.year = d2.year & d1.month = d2.month & d1.day = d2.day &
        d1.hour < d2.hour
        | d1.year = d2.year & d1.month = d2.month & d1.day = d2.day &
        d1.hour = d2.hour & d1.minute < d2.minute
-- Note 12/31/99 < 01/01/00
-- < is a total ordering on any time interval less than 50 years long
-- but it is not transitive on longer intervals
MESSAGE "<="(d1 d2:time)
        REPLY (b: boolean)
        WHERE b <=> d1 < d2 | d1 = d2
```

CONCEPT none: time

-- constant representing absence of a deadline constraint.

END

B. DESIGN DATABASE SCHEMA

1. Class Step

step.h **********

```
#ifndef __STEP_H
#define __STEP_H

#include <Object.h>
#include <Type.h>
#include <List.h>
#include <Reference.h>
#include <stream.h>

extern "C"
{
#include <sys/time.h>
#include <sys/types.h>
#include <string.h>
}

#ifndef __COMPONENT_H
#include "component.h"
#endif
#include "support_classes.h"


class COMP_REFERENCE:public Object
{
private:
          char*    priv_name;
          int      version_no;
          int      variation_no;

public:

          COMP_REFERENCE(APL *theAPL);
          virtual Type *getDirectType();
```

```
        COMP_REFERENCE();

        void set_priv_name(char* name){
                priv_name=name;
        }
        char* get_priv_name(){
                return priv_name;
        }

        void set_version_no(int value){
                version_no = value;
        }
        int  get_version_no(){
                return version_no;
        }

        void set_variation_no(int value){
                variation_no = value;
        }
        int  get_variation_no(){
                return variation_no;
        }

        void displaycomp();
};


class STEP : public Object
{
private :
        int       step_type;
        int       step_id;
        int       indicator;
        int       in_degree;
        int       estimated_duration;
        int       priority;
        char*     designer;
        int       status;
        int       required_expertise_level;
        Time      deadline;
        Time      start_time;
        Time      finish_time;
        time_t    date_created;
        time_t    date_of_current_status;
        Reference base_version;
```

```
        Reference  primary_input_list;
        Reference  secondary_input_list;
        Reference  output_list;
        Reference  affected_module_list;
        Reference  substep_list;
        Reference  part_of;
        Reference  preceded_by_list;

public:

        STEP(APL *theAPL);
        STEP(char* name, int s_id);
        virtual void putObject(OC_Boolean
                  deallocate=FALSE);
        virtual void deleteObject(OC_Boolean
                  deallocate=FALSE);
        virtual void Destroy(Boolean abort = FALSE);
        virtual Type *getDirectType();

        int get_step_type(){return step_type;}
        void set_step_type(int type){step_type =type;}
        void displayStep_id();
        int get_step_id(){
                  return step_id;
        }

        int get_indicator(){
                  return indicator;
        }
        void set_indicator(int type){
                  indicator =type;
        }

        int get_in_degree(){
                  return in_degree;
        }
        void set_in_degree(int type){
                  in_degree =type;
        }

        void set_estimated_duration(int value){
                  estimated_duration= value;
        }
        int get_estimated_duration(){
                  return estimated_duration;
        }
```

```
void set_priority(int value){
          priority= value;
}
int get_priority(){
          return priority;
}

void set_designer(char* a_name){
          designer=a_name;
}
char* get_designer(){
          return designer;
}

void  set_status(int value){
          status = value;
}
int get_status(){
          return status;
}

void  set_required_expertise_level(int value){
          required_expertise_level=value;
}
int get_required_expertise_level(){
          return required_expertise_level;
}

void set_deadline(Time value){
          deadline = value;
}
Time get_deadline(){
          return deadline;
}

void set_start_time(Time value){
          start_time = value;
}
Time get_start_time(){
          return start_time;
}

void set_finish_time(Time value){
          finish_time = value;
}
```

```
Time get_finish_time(){
          return finish_time;
}

// set creation time
    time_t setCreationDate();

// get creation time
    time_t getCreationDate(){
          return date_created;
}

time_t get_date_of_current_status(){
          return date_of_current_status;
}

void set_date_of_current_status(){
          date_of_current_status =
                    setCreationDate();
}

// get a list of primary input.
    List* primary_input(){
          return (List*)
          primary_input_list.Binding(this);
}

// reset a list of  primary input
    void  primary_input( List* parts) {
          primary_input_list.Reset(parts, this);
}


// get a list of secondary_input.
    List* secondary_input(){
          return (List*)
          secondary_input_list.Binding(this);
}

// reset a list of  secondary input
    void  secondary_input( List* parts) {
          secondary_input_list.Reset(parts, this);
}

// get a list of output
    List* output(){
```

```
            return (List*)output_list.Binding(this);
}

// reset a list of output
    void  output( List* parts) {
            output_list.Reset(parts, this);
}

// get a list of affected_modules
    List* affected_module(){
            return (List*)

            affected_module_list.Binding(this);
}

// reset a list of affected_modules
    void  affected_module( List* parts) {
            affected_module_list.Reset(parts, this);
}

// get a list of substeps
    List* substep(){
            return (List*)
                        substep_list.Binding(this);
}

// reset a list of substeps
    void  substep( List* parts) {
            substep_list.Reset(parts, this);
}

// get a list of preceded_by
    List* preceded_by(){
            return (List*)
            preceded_by_list.Binding(this);
}

// reset a list of preceded_by
    void  preceded_by( List* parts) {
            preceded_by_list.Reset(parts, this);
}


void set_parent_step(STEP* otherstep){
            part_of.Reset(otherstep,this) ;
}
```

```
        void set_base_version(COMP_REFERENCE* my_comp){
                base_version.Reset(my_comp,this);
        }
        COMP_REFERENCE* get_base_version();
        void add_substep(STEP* my_step);
        void add_predecessor(STEP* my_step);
        //void add_successor(STEP* my_step);
        void add_primary_input(COMP_REFERENCE* my_comp);
        void delete_primary_input(char* comp_name);
        void add_secondary_input(COMP_REFERENCE* my_comp);
        void delete_secondary_input(char* comp_name);
        void add_output(COMPONENT* my_comp);
        void add_affected_modules(COMP_REFERENCE*
                                                my_comp);
        void delete_affected_modules(char* comp_name);
        void show_primary_input();
        void show_secondary_input();
        void show_affected_modules();
        void show_output();
        void show_substeps();
        void show_preceding_steps();
        STEP* get_parent_step();
};

#endif  // __STEP_H
```

step.cxx **********

```
#ifndef __DDBDEFINES_H
#include "ddbdefines.h"
#endif

#include <Type.h>
#include <Object.h>
#include <GlobalEntities.h>
#include <Database.h>
#include <Directory.h>

extern "C"
{
        char *getenv(const char *);
#include <strings.h>
#include <ctype.h>
#include <stddef.h>
#include <string.h>
```

```
}

#ifndef __COMPONENT_H
#include "component.h"
#endif

#ifndef __STEP_H
#include "step.h"
#endif



//
============================================================
STEP::STEP(APL *theAPL):Object(theAPL),
deadline((APL*)0),start_time((APL*)0),finish_time((APL*)0)
{
}

STEP::STEP(char* name, int s_id):
Object(name),deadline(0,0,0,0,0), start_time(0,0,0,0,0),
finish_time(0,0,0,0,0)
{
          initDirectType((Type*)OC_lookup("STEP"));
          step_type = 0;
          indicator = 0;
          in_degree = 0;
          step_id = s_id;
          estimated_duration = 0;
          priority = 0;
          designer =(char*)0;
          status = 0;
          required_expertise_level = 0;
          date_created = setCreationDate();
          date_of_current_status = 0;
          base_version.initToNull();
          part_of.initToNull();
          primary_input_list.Init(new
          List((Type*)OC_lookup("COMP_REFERENCE")),this);
          secondary_input_list.Init(new
          List((Type*)OC_lookup("COMP_REFERENCE")),this);
          output_list.Init(new
          List((Type*)OC_lookup("COMPONENT")),this);
          affected_module_list.Init(new
          List((Type*)OC_lookup("COMP_REFERENCE")),this);
          substep_list.Init(new
```

```
            List((Type*)OC_lookup("STEP")),this);
            preceded_by_list.Init(new
            List((Type*)OC_lookup("STEP")),this);
}

Type *STEP::getDirectType()
{
            return (Type*)OC_lookup("STEP");
}

void STEP::putObject(OC_Boolean deallocate)
{
            //saves structure of the component lists
                ((List*)primary_input_list.Binding(this))-
>putObject(FALSE);
            ((List*)secondary_input_list.Binding(this))-
>putObject(FALSE);
            ((List*)output_list.Binding(this))-
>putObject(FALSE);
            ((List*)affected_module_list.Binding(this))-
>putObject(FALSE);
            ((List*)substep_list.Binding(this))-
>putObject(FALSE);
            ((List*)preceded_by_list.Binding(this))-
>putObject(FALSE);
            //  ((List*)successor_list.Binding(this))-
>putObject(FALSE);
            // saves the component itself
Object:
:
                putObject(deallocate);
}

void STEP::deleteObject(OC_Boolean deallocate)
{
            //deletes structure of the component lists
                ((List*)primary_input_list.Binding(this))-
>deleteObject(deallocate);
            ((List*)secondary_input_list.Binding(this))-
>deleteObject(deallocate);
            ((List*)output_list.Binding(this))-
>deleteObject(deallocate);
            ((List*)affected_module_list.Binding(this))-
>deleteObject(deallocate);
            ((List*)substep_list.Binding(this))-
>deleteObject(deallocate);
```

```
          ((List*)preceded_by_list.Binding(this))-
>deleteObject(deallocate);
          //  ((List*)successor_list.Binding(this))-
>deleteObject(deallocate);
          // deletes the component itself
Object:
:
              deleteObject(deallocate);
}

void STEP::Destroy(Boolean aborted)
{
          Entity* ent;
          ent = primary_input_list.Binding(this);
          delete ent;
          ent = secondary_input_list.Binding(this);
          delete ent;
          ent = output_list.Binding(this);
          delete ent;
          ent = affected_module_list.Binding(this);
          delete ent;
          ent = substep_list.Binding(this);
          delete ent;
          ent = preceded_by_list.Binding(this);
          delete ent;
          // ent = successor_list.Binding(this);
          delete ent;
          if (aborted) Object:
:
          Destroy(aborted);
}

// set creation time
time_t STEP::setCreationDate()
{
          time_t *mytloc =0;
          time_t theTime;
          return theTime = time(mytloc);
}

void STEP::add_substep(STEP* otherstep)
{
          List *child_nodes = (List
*)substep_list.Binding(this);

          if (!this)
```

```
        {
        cout << "<ERROR: cannot add a substep to a null
step\n";
                return;
        }
        if (!child_nodes)
        {
        cout << "<ERROR: cannot add a null substep to a
step\n";
                return;
        }

        child_nodes->Insert(otherstep);
        child_nodes->putObject();
        putObject();
}

void STEP::add_predecessor(STEP* otherstep)
{
        List *the_nodes = (List
*)preceded_by_list.Binding(this);

        if (!this)
        {
        cout << "<ERROR: cannot add a substep to a null step
\n";
                return;
        }
        if (!the_nodes)
        {
        cout << "<ERROR: cannot add a null substep  to a
step\n";
                return;
        }
        this->set_in_degree(this->get_in_degree()+1);
        the_nodes->Insert(otherstep);
        the_nodes->putObject();
        putObject();
}


void STEP::add_primary_input(COMP_REFERENCE* my_comp)
{
        List *the_nodes = (List
*)primary_input_list.Binding(this);
```

```
        if (!this)
        {
                cout << "<ERROR: cannot add a
COMP_REFERENCE to a null step\n";
                return;
        }
        if (!the_nodes)
        {
                cout << "<ERROR: cannot add a null
COMP_REFERENCE to a step\n";
                return;
        }

        the_nodes->Insert(my_comp);
        the_nodes->putObject();
        //        putObject();
}

void STEP::add_secondary_input(COMP_REFERENCE* my_comp)
{
        List *the_nodes = (List
*)secondary_input_list.Binding(this);

        if (!this)
        {
                cout << "<ERROR: cannot add a
COMP_REFERENCE to a null step\n";
                return;
        }
        if (!the_nodes)
        {
                cout << "<ERROR: cannot add a null
COMP_REFERENCE to a step\n";
                return;
        }

        the_nodes->Insert(my_comp);
        the_nodes->putObject();
        //       putObject();
}

void STEP::add_output(COMPONENT* my_comp)
{
        List *the_nodes = (List
*)output_list.Binding(this);
```

```
        if (!this)
        {
                cout << "<ERROR: cannot add a COMPONENT
to a null step\n";
                return;
        }
        if (!the_nodes)
        {
                cout << "<ERROR: cannot add a null
COMPONENT to a step\n";
                return;
        }

        the_nodes->Insert(my_comp);
        the_nodes->putObject();
        //      putObject();
}

void STEP::add_affected_modules(COMP_REFERENCE* my_comp)
{
        List *the_nodes = (List
*)affected_module_list.Binding(this);

        if (!this)
        {
                cout << "<ERROR: cannot add a
COMP_REFERENCE to a null step\n";
                return;
        }
        if (!the_nodes)
        {
                cout << "<ERROR: cannot add a null
COMP_REFERENCE to a step\n";
                return;
        }

        the_nodes->Insert(my_comp);
        the_nodes->putObject();
        //      putObject();
}

void STEP::delete_primary_input(char* my_comp)
{
        OC_Boolean FOUND=FALSE;
        List *my_list =
(List*)primary_input_list.Binding(this);
```

```
        ListIterator my_iterator(my_list);
        COMP_REFERENCE *the_comp;
        while(my_iterator.moreData()&& !FOUND)
        {
                the_comp =
(COMP_REFERENCE*)(Entity*)my_iterator();
                if(strcmp(the_comp ->get_priv_name(),
my_comp)==0){
                        my_list->Remove(my_list-
>Index(the_comp));
                        FOUND = TRUE;
                }
        }
}

void STEP::delete_secondary_input(char* my_comp)
{
        OC_Boolean FOUND=FALSE;
        List *my_list =
(List*)secondary_input_list.Binding(this);
        ListIterator my_iterator(my_list);
        COMP_REFERENCE *the_comp;
        while(my_iterator.moreData()&& !FOUND)
        {
                the_comp =
                (COMP_REFERENCE*)(Entity*)my_iterator();
                if(strcmp(the_comp ->get_priv_name(),
                my_comp)==0){
                        my_list->Remove(my_list-
                        >Index(the_comp));
                        FOUND = TRUE;
                }
        }
}


void STEP::delete_affected_modules(char* my_comp)
{
        OC_Boolean FOUND=FALSE;
        List *my_list =
        (List*)affected_module_list.Binding(this);
        ListIterator my_iterator(my_list);
        COMP_REFERENCE *the_comp;
        while(my_iterator.moreData()&& !FOUND)
        {
                the_comp =
```

```
                    (COMP_REFERENCE*)(Entity*)my_iterator();
                    if(strcmp(the_comp ->get_priv_name(),
                    my_comp)==0){
                              my_list->Remove(my_list-
                              >Index(the_comp));
                              FOUND = TRUE;
                    }
          }
}

COMP_REFERENCE* STEP::get_base_version()
{
          return
(COMP_REFERENCE*)base_version.Binding(this);
}

STEP* STEP::get_parent_step()
{
          return (STEP*)part_of.Binding(this);
}

void STEP::show_primary_input()
{
          List *my_list =
          (List*)primary_input_list.Binding(this);
          ListIterator my_iterator(my_list);
          COMP_REFERENCE *the_comp;
          //  cout<< "Primary inputs:              ";

          while(my_iterator.moreData())
          {
                    the_comp =
                    (COMP_REFERENCE*)(Entity*)my_iterator();
                    the_comp -> displaycomp();
          }
}


void STEP::show_secondary_input()
{
          List *my_list =
          (List*)secondary_input_list.Binding(this);
          ListIterator my_iterator(my_list);
          COMP_REFERENCE *the_comp;
          //  cout<< my_list->Cardinality() <<"\n";
          while(my_iterator.moreData())
```

```
        {
                the_comp =
                (COMP_REFERENCE*)(Entity*)my_iterator();
                the_comp -> displaycomp();
        }
}

void STEP::show_affected_modules()
{
        List *my_list =
        (List*)affected_module_list.Binding(this);
        ListIterator my_iterator(my_list);
        COMP_REFERENCE *the_comp;
        //  cout<< my_list->Cardinality() <<"\n";
        while(my_iterator.moreData())
        {
                the_comp =
                (COMP_REFERENCE*)(Entity*)my_iterator();
                the_comp -> displaycomp();
        }
}

void STEP::show_output()
{
        List *my_list = (List*)output_list.Binding(this);
        ListIterator my_iterator(my_list);
        COMPONENT *the_comp;
        //  cout<< "\nstep outputs:                ";

        while(my_iterator.moreData())
        {
                the_comp =
                (COMPONENT*)(Entity*)my_iterator();
                cout <<the_comp ->CompnentName();
        }
}

void STEP::show_substeps()
{
        List *my_list = (List*)substep_list.Binding(this);
        ListIterator my_iterator(my_list);
        STEP *the_step;
        //  cout<< my_list->Cardinality() <<"\n";
        while(my_iterator.moreData())
        {
                the_step =
```

```
(STEP*)(Entity*)my_iterator();
                    the_step-> displayStep_id();
          }
}

void STEP::show_preceding_steps()
{
          List *my_list =
(List*)preceded_by_list.Binding(this);
          ListIterator my_iterator(my_list);
          STEP *the_step;
          //  cout<< "\nstep predecessors:           ";
          int i=0;
          //  cout<< my_list->Cardinality() <<"\n";
          while(my_iterator.moreData())
          {
                    i = i+1;
                    the_step =
(STEP*)(Entity*)my_iterator();
                    cout <<the_step-> get_step_id();
                    if (i< my_list->Cardinality())
                              cout <<",";

          }
}


void STEP::displayStep_id()
{
          cout << step_id <<"    ";
}

COMP_REFERENCE::COMP_REFERENCE(APL *theAPL):Object(theAPL)
{
}

COMP_REFERENCE::COMP_REFERENCE()
{

initDirectType((Type*)OC_lookup("COMP_REFERENCE"));
          priv_name=(char*)0;
          version_no=0;
          variation_no=0;
}

Type* COMP_REFERENCE::getDirectType()
```

```
{
        return (Type*)OC_lookup("COMP_REFERENCE");
}

void COMP_REFERENCE::displaycomp()
{
        cout <<priv_name <<"\n";
}
```

step_Operations.h **********

```
#ifndef __STEP_OPERATIONS_H
#define __STEP_OPERATIONS_H

#include "component.h"
#include "step.h"
#include "support_classes.h"
#include "text_object.h"

char* get_red_of_extension(char* comp_name);
void create_step(char* dbname,char* protoname, char* comp_name);
void show_step(char* dbname,int step_id);
void show_steps(char* dbname,char* aName);
void Add_primary_input(int step_id,char* comp_name);
void Delete_primary_input(int step_id,char* comp_name);
void Add_secondary_input(int step_id,char* comp_name);
void Delete_secondary_input(int step_id,char* comp_name);
void Add_affected_modules(int step_id,char* comp_name);
void Delete_affected_modules(int step_id,char* comp_name);
void Update_precedence(int step_id,int preceding_step_id);
void Update_priority(int step_id,int value);
void Update_expertise_level(int step_id,int value);
void Update_deadline(int step_id,char* theDate);
void Update_estimated_duration(int step_id,int value);
```

```
        void Update_start_time(char* dbname,int step_id,char* theDate);
        void Update_finish_time(char* dbname,int step_id,char* theDate);
        void Update_status(char* dbname,int step_id,int value);
        void Update_designer(char* dbname,int step_id,char* aName);
        void get_scheduling_data(char* dbname, int step_id, char* curr_time);
        void get_Sched_data_1(char *dbname,char* listName,char* cur_time,char*
d_name);
        void get_scheduling_data_2(char* dbname, char* curr_time, char* d_name);
        void get_commit_data(char* dbname, int step_id);
        void set_secondary_input(STEP* a_step,char* compPath);
        void set_affected_modules(STEP* a_step, char* protoName,char* comp_name);
        void create_substep(char* dbname,int step_id,char* p_input,
                                        char* d_name,char* theDate, int duration);
        STEP* create_substep(STEP* the_step, char* p_input);
        void auto_create_substeps(char* dbname,int step_id);
        void commit_step(char* dbname,int step_id);
        void commit_substep(char* dbname,int step_id);
        void remove_step_from_schedule(char* dbname,char* step_id,char* myDate);
        void remove_step_from_sched(int step_id);
        void Update_Step(char* dbname,int step_id,char* theDate,
                        char* p_inputA, char* p_inputD,
                        char* s_inputA, char* s_inputD,
                        char* a_inputA, char* a_inputD,
                        int pri_value, int prec_vlaue,
                        int dur_vlaue, int exp_levvel);
        void step_update(int step_id,char* theDate,
                        char* p_inputA, char* p_inputD,
                        char* s_inputA, char* s_inputD,
                        char* a_inputA, char* a_inputD,
```

```
                    int pri_value, int prec_vlaue,
                    int dur_vlaue, int exp_level);
    void save_step_old_values(STEP* a_step);
    void Undo_step_update(char* dbname,int step_id);
    void dump_step_components(char* dbname, int step_id);
    void find_assigned_step(char* dbname, char* user);
    void suspend_abandon_step(char* dbname,int step_id, int new_status);
    void update_base_versions(int step_id,char* temp1);
    void Update_deadline(char* dbname,int step_id);
    void Early_warning(char* dbname, char* myTime);

    #endif // __STEP_OPERATIONS_H
```

step_Operations.cxx **********

```
#include <Database.h>
#include <Directory.h>
#include <string.h>
#include <Set.h>
#include <stdlib.h>
#include "My_String.h"
#include "component.h"
#include "step.h"
#include "support_classes.h"
#include "text_object.h"
#include "comp_Operations.h"
#include "step_Operations.h"
#include "sched.h"
#include "schedOp.h"
#include "person.h"

static char *Level[3] = {"Low", "Medium", "High"};
static char *Status[6] =
{"proposed","approved","scheduled","assigned","completed","a
bandoned"};

extern char *thepath;
extern int  warning_time;
extern char *dirNamePtr;
```

```
extern char *DESIGN_DATABASE_DIRECTORY;

char* get_red_of_extension(char* comp_name)
{
        char my_word[128];
        char *word[7];
        char* separator=".";
        word[0]=strtok(comp_name,separator);
        int i=0;
        while(word[i] != NULL){
              i=i+1;
              word[i]=strtok(NULL,separator);
        }
        int k = i-3;
        if(k==0) return word[0];
        else{
            if(k==1){
              sprintf(my_word,"%s%s%s",word[0],".",word[1]);
               return my_word;
            }
            else {
                 if(k==2){

sprintf(my_word,"%s%s%s%s%s",word[0],".",word[1],".",word[2]
);
                 return my_word;
                 }
                 else{

sprintf(my_word,"%s%s%s%s%s%s%s",word[0],".",word[1],
                                    ".",word[2],".",word[3]);
                    return my_word;
                    }
                 }
           }
}

STEP* find_step(int step_id)
{
  OC_Boolean FOUND=FALSE;
  Set *aSet = (Set *)OC_lookup("step_set");

  // Abort if there is no set
  if ( aSet == NULL ) {
      cout << "there is no steps in database yet.\n";
      return NULL;
```

```
  }

//  cout << aSet->Name() << " has " << aSet->Cardinality() <<
" items.\n\n";

  // Ask the set object for an iterator
  Iterator* anIterator = aSet->getIterator();
  STEP* the_step;
  // For each item in the iterator
  while(anIterator->moreData()&& !FOUND) {
      // Get the item
      the_step=(STEP*)(Entity *)(anIterator->operator()()) ;
        if(the_step->get_step_id()==step_id)
         FOUND = TRUE;
        }
  if(FOUND)
      return the_step;
  else
      return NULL;
}


void find_assigned_step(char* dbname, char* user)
{
  OC_open(dbname);
  OC_transactionStart();
  OC_Boolean FOUND = FALSE;
  Set *aSet = (Set *)OC_lookup("step_set");
  Iterator* anIterator = aSet->getIterator();
  STEP* the_step;
  // For each item in the iterator
  while(anIterator->moreData() && !FOUND) {
  the_step=(STEP*)(Entity *)(anIterator->operator()());
  if(the_step->get_designer() !=0){
  char* salah= new char[strlen(the_step->get_designer())+1];
  strcpy(salah,the_step->get_designer());
  if(strcmp(salah,user)==0 && the_step->get_status()==3 &&
the_step->get_indicator()==0){
       FOUND = TRUE;
       char my_name[16] ;
       sprintf(my_name,"%s%s%d",".","step_",the_step-
>get_step_id());
       cout << "F" << "\n";
       cout << my_name << "\n";
       cout << the_step->get_step_id() << "\n";
       }
```

```
      }
   }
 if(!FOUND) cout << "N" << "\n";
 OC_transactionCommit();
 OC_close();
}

void dump_step_components(char* dbname,int step_id)
{
  OC_open(dbname);
  OC_transactionStart();
    char my_name[8] ;
    char protoname[64];
    STEP* the_step= find_step(step_id);
    sprintf(my_name,"%d",the_step->get_step_id());
    My_String
temp=My_String(".")+My_String("step_")+My_String(my_name);
    dirNamePtr =(char*) temp;
    COMP_REFERENCE* my_comp= the_step->get_base_version();
       char *temp2=my_comp->get_priv_name();
       int var=my_comp->get_variation_no();
       int ver=my_comp->get_version_no();
       sprintf(protoname,"%s %d:%d", temp2,var, ver);

       List *my_list=the_step->primary_input();
       COMP_REFERENCE* my_ref =(COMP_REFERENCE*)my_list-
>getEntityElement(0);
       char* check = my_ref->get_priv_name()+strlen(my_ref-
>get_priv_name())-10;
       if(strcmp(check,".spec.psdl")==0){
       My_String temp=My_String(my_ref->get_priv_name())-10;
       char* comp_name =(char*) temp;
       DumpComponent(protoname,comp_name);
       }
       else{
        My_String temp=My_String(my_ref->get_priv_name())-9;
         char* comp_name =(char*) temp;
         DumpComponent(protoname,comp_name);
         List *my_list=the_step->secondary_input();
         ListIterator my_iterator(my_list);
         while(my_iterator.moreData())
         {
          COMP_REFERENCE*
my_comp=(COMP_REFERENCE*)(Entity*)my_iterator();
         My_String temp=My_String(my_comp->get_priv_name())-
10;
```

```
            char* comp_name =(char*) temp;
            Dump_Spec_File1(protoname,comp_name);
           }
         }
 the_step->set_indicator(1);
 the_step->putObject();
 OC_transactionCommit();
 OC_close();
}

void Early_warning(char* dbname, char* myTime)
{
  OC_open(dbname);
  OC_transactionStart();
  int my_id;
  Time T1(myTime);
  List *aList = (List *)OC_lookup("MySchedule");
  if(aList  != NULL) {
     Iterator* my_iterator = aList->getIterator();
     // For each item in the iterator
     while(my_iterator->moreData()) {
     Schedule* nextAssignment = (Schedule*)
(Entity*)((*my_iterator)());
     if( nextAssignment->AssignmentFinish()- T1 <=
warning_time)
      {
        sscanf(nextAssignment->Name(),"%d", &my_id);
        STEP* the_step=find_step(my_id);
        if(the_step->get_status()== 3 && the_step-
>get_indicator() != 2)
           {
           cout << the_step->get_step_id()<< "\n";
           cout << the_step->get_designer() << "\n";
             the_step->set_indicator(2);
           the_step->putObject();
             }
      }
     }
  }
 OC_transactionCommit();
 OC_close();
}

void show_step(char* dbname, int step_id)
{
      OC_open(dbname);
```

```
      OC_transactionStart();
      STEP* the_step=find_step(step_id) ;
        if(the_step != NULL){
        COMP_REFERENCE* my_comp = the_step-
>get_base_version();
        cout <<my_comp->get_priv_name() <<" "
               << my_comp->get_variation_no()
               <<" " << my_comp->get_version_no() <<"\n";
        cout   <<the_step->get_estimated_duration() <<"\n";
        cout   <<the_step->get_priority() << "\n";
        cout   <<Level[the_step ->
get_required_expertise_level()]
               << "\n";
        cout   <<Status[the_step->get_status()]<< "\n";
        cout   <<the_step->get_designer() << "\n";
      cout  <<the_step->get_deadline().makeString() << "\n";
        cout   <<the_step->get_start_time().makeString() <<
"\n";
       cout   <<the_step->get_finish_time().makeString() <<
"\n";
        the_step->show_primary_input();
        List* my_list=the_step->secondary_input();
        cout   << my_list->Cardinality() << "\n";
        the_step->show_secondary_input();
        my_list=the_step->affected_module();
        cout   << my_list->Cardinality() << "\n";
        the_step->show_affected_modules();
        my_list=the_step->substep();
        int n =  my_list->Cardinality();
        cout << my_list->Cardinality() << "\n";
        the_step->show_substeps();
        my_list=the_step->preceded_by();
        if (n==0)
        cout <<my_list->Cardinality() << "\n";
        else
        cout <<  "\n" <<my_list->Cardinality() << "\n";
        the_step->show_preceding_steps();
        cout <<  "\n" <<  "\n";
          }
 OC_transactionCommit();
 OC_close();
}

void get_commit_data(char* dbname, int step_id)
{
        OC_open(dbname);
```

```
        OC_transactionStart();
        STEP* the_step=find_step(step_id);
        if(the_step != NULL){
         the_step->get_base_version()->displaycomp();
         the_step->show_primary_input();
         the_step->show_output();
        }
 OC_transactionCommit();
 OC_close();
}

void get_scheduling_data_2(char* dbname, char*
curr_time,char* d_name)
{
  OC_open(dbname);
  OC_transactionStart();
  int my_id;
  Time T1(curr_time);
  Time T(0,0,0,0,0);
  List *aList = (List *)OC_lookup("MySchedule");
  if(aList  != NULL) {
     Iterator* my_iterator = aList->getIterator();
     // For each item in the iterator
     while(my_iterator->moreData()) {
     Schedule* nextAssignment = (Schedule*)
(Entity*)((*my_iterator)());
        sscanf(nextAssignment->Name(),"%d", &my_id);
        STEP* the_step=find_step(my_id);
           if(the_step->get_status()==2 ||
                    (the_step->get_status()==3 &&
strcmp(the_step->get_designer(),d_name)==0)){
           cout << the_step->get_step_id()<< "\n";
           int T2= the_step->get_deadline() - T1;
           if (T2 < 0)
           cout <<"1000" << "\n";
           else
           cout <<T2 << "\n";
           cout <<the_step->get_priority() << "\n";
           cout <<the_step->get_estimated_duration() <<"\n";
           cout <<"{";
           the_step->show_preceding_steps();
           cout <<"} \n";
           cout <<Level[the_step-
>get_required_expertise_level()]<< "\n";
           cout << the_step->get_in_degree() << "\n";
        }
```

```
      }
    }
 OC_transactionCommit();
 OC_close();
}



void get_scheduling_data(char* dbname, int step_id, char*
curr_time)
{
  OC_open(dbname);
  OC_transactionStart();
  int my_id;
  Time T1(curr_time);
  Time T(0,0,0,0,0);
  List *aList = (List *)OC_lookup("MySchedule");
  if(aList  != NULL) {
     Iterator* my_iterator = aList->getIterator();
     // For each item in the iterator
     while(my_iterator->moreData()) {
     Schedule* nextAssignment = (Schedule*)
(Entity*)((*my_iterator)());
        sscanf(nextAssignment->Name(),"%d", &my_id);
        STEP* the_step=find_step(my_id);
           if(the_step->get_status()==2){
           cout << the_step->get_step_id()<< "\n";
           int T2= the_step->get_deadline() - T1;
           if (T2 < 0)
           cout <<"1000" << "\n";
           else
           cout <<T2 << "\n";
           cout <<the_step->get_priority() << "\n";
           cout <<the_step->get_estimated_duration() <<"\n";
           cout <<"{";
           the_step->show_preceding_steps();
           cout <<"} \n";
           cout <<Level[the_step-
>get_required_expertise_level()]
                << "\n";
           cout << the_step->get_in_degree() << "\n";
        }
      }
    }
   if(step_id !=0)
     {
```

```
      STEP* a_step=find_step(step_id);
      if(a_step != NULL){
     if(a_step->get_status() != 1 || a_step->get_step_type()
!= 0)
        cout << "Cannot schedule None approved step or part
of a top step" << "\n";
      else{
      List* my_list=a_step->substep();
      ListIterator my_iterator(my_list);
      STEP *the_step;
        while(my_iterator.moreData())
      {
         the_step = (STEP*)(Entity*)my_iterator();
           cout << the_step->get_step_id()<< "\n";
           int T2= the_step->get_deadline() - T1;
           if (T2 < 0)
           cout <<"1000" << "\n";
           else
           cout <<T2 << "\n";
           cout <<the_step->get_priority() << "\n";
           cout <<the_step->get_estimated_duration() <<"\n";
           cout <<"{";
           the_step->show_preceding_steps();
           cout <<"} \n";
           cout <<Level[the_step-
>get_required_expertise_level()]
                << "\n";
           cout << the_step->get_in_degree() << "\n";
      }
      }
    }
   }
 OC_transactionCommit();
 OC_close();
}

void get_Sched_data_1(char *dbname,char* listName,char*
cur_time,char* d_name)
{
  OC_open(dbname);
  OC_transactionStart();
  int my_id;
  List *aList = (List *)OC_lookup(listName);
  if(aList  == NULL) {
   //   cout << "No Such Schedule ...\n";
  OC_transactionCommit();
```

```
  OC_close();
  return;
    }
   Time T1(cur_time);

   Iterator* anIterator = aList->getIterator();

  // For each item in the iterator
  while(anIterator->moreData()) {
    Schedule* nextAssignment = (Schedule*)
(Entity*)((*anIterator)());
        sscanf(nextAssignment->Name(),"%d", &my_id);
        STEP* the_step=find_step(my_id);
        if(the_step->get_status()==3 && strcmp(
          the_step->get_designer(),d_name) !=0){
        cout << nextAssignment->Name() <<"\n";
        int T3=nextAssignment-> AssignmentFinish()-T1;
        cout <<T3 <<"\n";
        cout << nextAssignment->AssignedDesigner() << "\n";
       }
    }
  delete anIterator;
  OC_transactionCommit();
  OC_close();
}

void remove_step_from_schedule(char* dbname,char*
step_id,char* myDate)
{
  OC_open(dbname);
  OC_transactionStart();
  deleteAssignment1("MySchedule", step_id);
  Time T(myDate);
  int my_step_id, an_id;
  sscanf(step_id,"%d",&an_id);
  STEP* the_step=find_step(an_id);
  char* dname=the_step->get_designer();
  Person* aPerson=(Person*)OC_lookup(dname);
  if(aPerson != NULL){
  aPerson->PersonStatus(0);
  aPerson->putObject();
  }
  the_step->set_finish_time(T);
  the_step->putObject();
  Time T2 = the_step->get_start_time()+the_step-
>get_estimated_duration();
```

```
  if((T2 - T)==0)
  cout << "N" <<"\n";
  else
   cout <<"R" <<"\n";

  List *aList = (List *)OC_lookup("MySchedule");
  if(aList  == NULL) {
//      cout << "No Such Schedule ...\n";
  OC_transactionCommit();
  OC_close();
  return;
    }

   Iterator* anIterator = aList->getIterator();
  // For each item in the iterator
  while(anIterator->moreData()) {
    Schedule* nextAssignment = (Schedule*)
(Entity*)((*anIterator)());
    sscanf(nextAssignment->Name(),"%d", &my_step_id);
    STEP* my_step=find_step(my_step_id);
    if(my_step != NULL){
       List* my_list = my_step->preceded_by();
       Iterator* my_Iterator = my_list->getIterator();
       while(my_Iterator->moreData()) {
            STEP* a_step=(STEP*)(Entity*)((*my_Iterator)());
            if(a_step->get_step_id()== an_id)
            my_step->set_in_degree(my_step->get_in_degree()-
1);
              my_step->putObject();
            }
       }
    if(my_step->get_in_degree()==0){
       if(strcmp(dname,nextAssignment-
>AssignedDesigner())==0){
         cout << my_step->get_step_id() <<"\n";
         cout << dname <<"\n";
         }
       else{
         Person* aPerson=(Person*)OC_lookup(
                    nextAssignment->AssignedDesigner());
         if(aPerson != NULL){
             if(aPerson->PersonStatus()==0){
              cout << my_step->get_step_id() <<"\n";
                cout << nextAssignment-
>AssignedDesigner()<<"\n";
              }
```

```
          }
        }
      }
    }
 OC_transactionCommit();
 OC_close();
}


void remove_step_from_sched(int step_id)
{
  char my_id[8];
  sprintf(my_id, "%d",step_id);
  int my_step_id;
  deleteAssignment1("MySchedule", my_id);
  STEP* the_step=find_step(step_id);
  if (the_step != NULL){
  char* dname=the_step->get_designer();
  if(the_step->get_status() == 3){
     Person* aPerson=(Person*)OC_lookup(dname);
     if(aPerson != NULL){
     aPerson->PersonStatus(0);
     aPerson->putObject();
     cout << step_id << "\n";
     cout << dname << "\n";
     }
   }
  List *aList = (List *)OC_lookup("MySchedule");
  if(aList  == NULL) {
//   cout << "No Such Schedule ...\n";
  return;
    }
  Iterator* anIterator = aList->getIterator();
  // For each item in the iterator
  while(anIterator->moreData()) {
    Schedule* nextAssignment = (Schedule*)
          (Entity*)((*anIterator)());
    sscanf(nextAssignment->Name(),"%d", &my_step_id);
    STEP* my_step=find_step(my_step_id);
    if(my_step != NULL){
       List* my_list = my_step->preceded_by();
       Iterator* my_Iterator = my_list->getIterator();
       while(my_Iterator->moreData()) {
            STEP* a_step=(STEP*)(Entity*)((*my_Iterator)());
            if(a_step->get_step_id()== step_id)
            my_step->set_in_degree(my_step->get_in_degree()-
```

```
1);
                my_step->putObject();
              }
        }
    }
   }
}


void suspend_abandon_step(char* dbname,int step_id, int
new_status)
{
    OC_open(dbname);
    OC_transactionStart();
    Time T(0,0,0,0,0);
    STEP* my_step=find_step(step_id);
    if(my_step != NULL){
       if(my_step->get_status() <= 1){
         if(my_step->get_step_type()==0){
           List* my_list=my_step->substep();
           ListIterator my_iterator(my_list);
           while(my_iterator.moreData()){
             STEP* a_step=(STEP*)(Entity*)my_iterator();
             a_step->set_status(new_status);
             a_step->putObject();
           }
          }
          my_step->set_status(new_status);
          my_step->putObject();
        }
       else
      if(my_step->get_status() == 2 || my_step->get_status()
== 3){
          if(my_step->get_step_type()==0){
              List* my_list=my_step->substep();
              ListIterator my_iterator(my_list);
              while(my_iterator.moreData()){
                 STEP* a_step=(STEP*)(Entity*)my_iterator();
                 remove_step_from_sched(a_step-
>get_step_id());
                 a_step->set_status(new_status);
                 a_step->set_start_time(T);
                 List* a_list=a_step->preceded_by();
               a_step->set_in_degree(a_list->Cardinality());
                 a_step->putObject();
               }
```

```
            my_step->set_status(new_status);
            my_step->set_start_time(T);
            my_step->putObject();
            }
          else{
            remove_step_from_sched(step_id);
            my_step->set_status(new_status);
            my_step->putObject();
              }
           }
      }
    OC_transactionCommit();
    OC_close();
}



void show_steps(char* dbname,char* aName)
{
        OC_open(dbname);
        OC_transactionStart();
  Set *aSet = (Set *)OC_lookup("step_set");

  // Abort if there is no set
  if ( aSet == NULL ) {
      cout << "there is no steps in database yet.\n";
      return;
  }

  cout << aSet->Name() << " has " << aSet->Cardinality() << "
items.\n\n";

  // Ask the set object for an iterator
  Iterator* anIterator = aSet->getIterator();

  // For each item in the iterator
  while(anIterator->moreData()) {

      // Get the item
      STEP* the_step=(STEP*)(Entity *)(anIterator-
>operator()()) ;
      // Print out its name and value
      if(strcmp(aName,"all")==0)
      cout << the_step->get_step_id() << ",  Status: "
            << Status[the_step->get_status()]<< "\n";
```

```
      else{
         if(strcmp(aName,"top")== 0){
           if(the_step->get_step_type()==0)
             cout << the_step->get_step_id() << ",  Status: "
                       << Status[the_step->get_status()]<<
"\n";
         }
         else
             if(strcmp(aName,Status[the_step-
>get_status()])==0)
               cout << the_step->get_step_id() << ",
Status: "
                    << Status[the_step->get_status()]<< "\n";
         }
  }

  // Be sure to cleanup heap based iterators
  delete anIterator;
 OC_transactionCommit();
 OC_close();
}

void Add_primary_input(int step_id,char* thename)
{
          if( thename[0] !='0'){
          char comp_name[64];
          int var, ver;
          sscanf(thename,"%s %d:%d",comp_name, &var, &ver);
          STEP* a_step=find_step(step_id);
          if(a_step !=NULL){
          COMP_REFERENCE* my_ref=new COMP_REFERENCE();
          my_ref->set_priv_name(comp_name);
          my_ref->set_version_no(ver);
          my_ref->set_variation_no(var);
          my_ref->putObject();
          a_step->add_primary_input(my_ref);
          a_step->putObject();
          }
          else
             cout <<"STEP: " << step_id <<"   is not in the
DDB\n";
        }
}

void Delete_primary_input(int step_id,char* comp_name)
{
```

```
        if( comp_name[0] !='0'){
           STEP* a_step=find_step(step_id);
           if(a_step !=NULL){
                    a_step->delete_primary_input(comp_name);
                 a_step->putObject();
                     }
         }
}
void Add_secondary_input(int step_id,char *thename)
{
        if( thename[0] !='0'){
           char comp_name[64];
           int var, ver;
           sscanf(thename,"%s %d:%d",comp_name, &var, &ver);
           STEP* a_step=find_step(step_id);
           if(a_step !=NULL){
           COMP_REFERENCE* my_ref=new COMP_REFERENCE();
           my_ref->set_priv_name(comp_name);
           my_ref->set_version_no(ver);
           my_ref->set_variation_no(var);
           my_ref->putObject();
           a_step->add_secondary_input(my_ref);
           a_step->putObject();
           }
           else
              cout <<"STEP: " << step_id <<"   is not in the
DDB\n";
           }
}

void Delete_secondary_input(int step_id,char* comp_name)
{
        if( comp_name[0] !='0'){
           STEP* a_step=find_step(step_id);
           if(a_step !=NULL){
                 a_step->delete_secondary_input(comp_name);
                 a_step->putObject();
                 }
           }
}


void Add_affected_modules(int step_id,char* thename)
{
        if( thename[0] !='0'){
           char comp_name[64];
```

```
            int var, ver;
            sscanf(thename,"%s %d:%d",comp_name, &var, &ver);
            STEP* a_step=find_step(step_id);
            if(a_step !=NULL){
            COMP_REFERENCE* my_ref=new COMP_REFERENCE();
            my_ref->set_priv_name(comp_name);
            my_ref->set_version_no(ver);
            my_ref->set_variation_no(var);
            my_ref->putObject();
            a_step->add_affected_modules(my_ref);
            a_step->putObject();
            }
            else
               cout <<"STEP: " << step_id <<"   is not in the
DDB\n";
            }
}
void Delete_affected_modules(int step_id,char* comp_name)
{
       if( comp_name[0] !='0'){
            STEP* a_step=find_step(step_id);
            if(a_step !=NULL){
                  a_step->delete_affected_modules(comp_name);
                  a_step->putObject();
                  }
            }
}


void Update_precedence(int step_id,int preceding_step_id)
{
 OC_Boolean FOUND = FALSE;
 if(preceding_step_id !=0){
 STEP* a_step=find_step(step_id);
 if(a_step !=NULL){
   STEP* the_step=find_step(preceding_step_id);
   if(the_step){
     List* the_list= a_step->preceded_by();
      ListIterator the_iterator(the_list);
         while(the_iterator.moreData())
         {
          STEP* step1=(STEP*)(Entity*)the_iterator();
          if (step1->get_step_id()== preceding_step_id)
             FOUND = TRUE;
          }
     if(!FOUND){
```

```
     a_step->add_predecessor(the_step);
     a_step->putObject();
     if(the_step->get_step_type()==0 && a_step-
>get_step_type()==0){
      COMP_REFERENCE* my_ref=the_step->get_base_version();
      COMP_REFERENCE* my_ref1=a_step->get_base_version();
      if(strcmp(my_ref->get_priv_name(), my_ref1-
>get_priv_name())==0 &&
         my_ref->get_variation_no()== my_ref1-
>get_variation_no() &&
         my_ref->get_version_no()== my_ref1-
>get_version_no()){
         List* my_list = the_step->substep();
           ListIterator my_iterator(my_list);
         while(my_iterator.moreData())
         {
          STEP* my_step=(STEP*)(Entity*)my_iterator();
          List* a_list = my_step->primary_input();
          COMP_REFERENCE* a_ref= (COMP_REFERENCE*)a_list-
>getEntityElement(0);
    cout <<"first: "<< a_ref->get_priv_name() << "\n";
          List* my_list1 = a_step->substep();
          ListIterator my_iterator1(my_list1);
          while(my_iterator1.moreData()){
            STEP* my_step1=(STEP*)(Entity*)my_iterator1();
            List* a_list1 = my_step1->primary_input();
           COMP_REFERENCE* a_ref1= (COMP_REFERENCE*)a_list1-
>getEntityElement(0);
     cout <<"second: "<< a_ref1->get_priv_name() << "\n";
            if(strcmp(a_ref->get_priv_name(), a_ref1-
>get_priv_name())==0){
            my_step1->add_predecessor(my_step);
            my_step1->putObject();
            }
            }
         }
        }
       }
      }
          }
          else
           cout <<"STEP: "<< preceding_step_id<<" is not in
the DDB\n";
          }
          else
                    cout <<"STEP: " << step_id <<"   is not
```

```
in the DDB\n";
          }
}

void Update_priority(int step_id,int value)
{
          if(value !=0){
          STEP* a_step=find_step(step_id);
          if(a_step !=NULL){
                    a_step->set_priority(value);
                    a_step->putObject();
          }
          else
                    cout <<"STEP: " << step_id <<"   is not
in the DDB\n";
          }
}

void Update_expertise_level(int step_id,int value)
{
          if(value !=5){
          STEP* a_step=find_step(step_id);
          if(a_step !=NULL){
                a_step->set_required_expertise_level(value);
                a_step->putObject();
          }
          else
              cout <<"STEP: " << step_id <<"   is not in the
DDB\n";
          }
}

void Update_status(char* dbname,int step_id,int value)
{
        OC_open(dbname);
        OC_transactionStart();
          STEP* a_step=find_step(step_id);
          if(a_step !=NULL){
                a_step->set_status(value);
                a_step->putObject();
                if (value == 2 || value == 3){
                   STEP* my_step= a_step->get_parent_step();
                    if(my_step!= NULL){
                       if (my_step->get_status()< value){
                       my_step->set_status(value);
                       my_step->putObject();
```

```
                        }
                    }
                }
            }
            else
                cout <<"STEP: " << step_id <<"   is not in the
DDB\n";
 OC_transactionCommit();
 OC_close();
}


void Update_deadline(int step_id,char* theDate)
{
        if(theDate[0] !=0  && theDate[1] !=0){
          STEP* a_step=find_step(step_id);
          if(a_step !=NULL){
                    Time theTime(theDate);
                    a_step->set_deadline(theTime);
                    a_step->putObject();
         List* my_list = a_step->substep();
         ListIterator an_iterator(my_list);
         while(an_iterator.moreData())
           {
            STEP* my_step=(STEP*)(Entity*)an_iterator();
            my_step->set_deadline(theTime);
            my_step->putObject();
          }
          }
          else
              cout <<"STEP: " << step_id <<"   is not in the
DDB\n";
          }
}

void Update_deadline(char* dbname,int step_id)
{
 OC_open(dbname);
 OC_transactionStart();
 OC_Boolean FOUND=FALSE;
 char an_id[8];
 Time T(0,0,0,0,0);
 sprintf(an_id,"%d",step_id);
 STEP* a_step=find_step(step_id);
 if(a_step !=NULL){
    List *aList = (List *)OC_lookup("MySchedule");
```

```
    if(aList  != NULL) {
       Iterator* my_iterator = aList->getIterator();
       // For each item in the iterator
       while(my_iterator->moreData()&& !FOUND) {
       Schedule* nextAssignment = (Schedule*)
(Entity*)((*my_iterator)());
       if( strcmp(nextAssignment->Name(),an_id)==0 ){
          FOUND = TRUE;
          T = nextAssignment->AssignmentFinish();
          }
        }
         STEP* the_step=a_step->get_parent_step();
         the_step->set_deadline(T);
         the_step->putObject();
         List* my_list = the_step->substep();
         ListIterator an_iterator(my_list);
         while(an_iterator.moreData())
           {
            STEP* my_step=(STEP*)(Entity*)an_iterator();
            my_step->set_deadline(T);
            my_step->putObject();
          }
        }
    }
 OC_transactionCommit();
 OC_close();
}


void Update_estimated_duration(int step_id,int value)
{
 OC_Boolean FOUND=FALSE;
 char an_id[8];
 if(value !=0){
 sprintf(an_id,"%d",step_id);
   STEP* a_step=find_step(step_id);
   if(a_step !=NULL){
      if(a_step->get_status()==3){
        int n = value - a_step->get_estimated_duration();
        List *aList = (List *)OC_lookup("MySchedule");
        if(aList  != NULL) {
        Iterator* my_iterator = aList->getIterator();
        // For each item in the iterator
        while(my_iterator->moreData()&& !FOUND) {
        Schedule* nextAssignment = (Schedule*)
(Entity*)((*my_iterator)());
```

```
        if( strcmp(nextAssignment->Name(),an_id)==0 ){
         FOUND = TRUE;
         nextAssignment->AssignmentFinish(nextAssignment-
>AssignmentFinish()+n);
           nextAssignment->putObject();
          }
       }
      }
     a_step->set_estimated_duration(value);
     a_step->putObject();
     }
     else
       cout <<"STEP: " << step_id <<"   is not in the DDB\n";
   }
}

void Update_start_time(char* dbname,int step_id,char*
theDate)
{
        OC_open(dbname);
        OC_transactionStart();
        Time T(0,0,0,0,0);
          STEP* a_step=find_step(step_id);
          if(a_step !=NULL){
                Time theTime(theDate);
                a_step->set_start_time(theTime);
                a_step->putObject();
                STEP* the_step=a_step->get_parent_step();
                if(strcmp(the_step-
>get_start_time().makeString(),
                                T.makeString()) ==0){
                   the_step->set_start_time(theTime);
                   cout << "done" << "\n";
                       the_step->putObject();
                       }
          }
          else
              cout <<"STEP: " << step_id <<"   is not in the
DDB\n";
 OC_transactionCommit();
 OC_close();
}

void Update_finish_time(char* dbname,int step_id,char*
theDate)
```

```
{
        OC_open(dbname);
        OC_transactionStart();
          STEP* a_step=find_step(step_id);
          if(a_step !=NULL){
                Time theTime(theDate);
                a_step->set_finish_time(theTime);
                a_step->putObject();
                STEP* the_step=a_step->get_parent_step();
                if(theTime > the_step->get_finish_time()){
                   the_step->set_finish_time(theTime);
                       the_step->putObject();
                       }
          }
          else
              cout <<"STEP: " << step_id <<"   is not in the
DDB\n";
 OC_transactionCommit();
 OC_close();
}

void Update_designer(char* dbname,int step_id,char* aName)
{
        OC_open(dbname);
        OC_transactionStart();
          STEP* a_step=find_step(step_id);
          if(a_step !=NULL){
                a_step->set_designer(aName);
                a_step->putObject();
          }
          else
              cout <<"STEP: " << step_id <<"   is not in the
DDB\n";
 OC_transactionCommit();
 OC_close();
}

void Update_Step(char* dbname,int step_id,char* theDate,
                 char* p_inputA, char* p_inputD,
                 char* s_inputA, char* s_inputD,
                 char* a_inputA, char* a_inputD,
                 int pri_value,  int   prec_vlaue,
                 int dur_vlaue,  int   exp_levvel)
{
        OC_open(dbname);
        OC_transactionStart();
```

```
        STEP* a_step=find_step(step_id);
        if(a_step !=NULL){
           if(a_step->get_status()<= 1){

step_update(step_id,theDate,p_inputA,p_inputD,s_inputA,s_inp
utD,

a_inputA,a_inputD,pri_value,prec_vlaue,dur_vlaue,exp_levvel)
;
                a_step->putObject();
                cout << "d" <<"\n";
                }
           else
              if(a_step->get_status()== 2 || a_step-
>get_status()== 3)
                {
//                 save_step_old_values(a_step);

step_update(step_id,theDate,p_inputA,p_inputD,s_inputA,s_inp
utD,

a_inputA,a_inputD,pri_value,prec_vlaue,dur_vlaue,exp_levvel)
;
                   cout << "s" <<"\n";
                     a_step->putObject();
                 }
              else
               if(a_step->get_status()== 4)
                 cout << "c" <<"\n";
               else
                 if(a_step->get_status()== 5)
                    cout << "a" <<"\n";

         }
          OC_transactionCommit();
          OC_close();
}

void step_update(int step_id,char* theDate,
                 char* p_inputA, char* p_inputD,
                 char* s_inputA, char* s_inputD,
                 char* a_inputA, char* a_inputD,
                 int pri_value,  int   prec_vlaue,
                 int dur_vlaue,  int   exp_level)
{
             Update_deadline(step_id,theDate);
```

```
            Add_primary_input(step_id, p_inputA);
            Delete_primary_input(step_id, p_inputD);
            Add_secondary_input(step_id,s_inputA);
            Delete_secondary_input(step_id,s_inputD);
              Add_affected_modules(step_id,a_inputA);
            Delete_affected_modules(step_id,a_inputD);
            Update_priority(step_id,pri_value);
              Update_precedence(step_id,prec_vlaue);
              Update_estimated_duration(step_id,dur_vlaue);
              Update_expertise_level(step_id, exp_level);
}

void save_step_old_values(STEP* a_step)
{
          STEP* temp_step= new STEP("temp_step",0);
        temp_step->primary_input(a_step->primary_input());
      temp_step->secondary_input(a_step->secondary_input());
          temp_step->affected_module(a_step-
>affected_module());
          temp_step->set_priority(a_step->get_priority());
          temp_step->preceded_by(a_step->preceded_by());
          temp_step->set_estimated_duration(a_step-
>get_estimated_duration());
          temp_step->set_deadline(a_step->get_deadline());
          temp_step->set_required_expertise_level(a_step-
>get_required_expertise_level());
          temp_step->putObject();
}

void Undo_step_update(char* dbname,int step_id)
{
          OC_open(dbname);
          OC_transactionStart();
          STEP* a_step=find_step(step_id);
        if(a_step){
           STEP* temp_step= (STEP*)OC_lookup("temp_step");
           if(temp_step){
          a_step->primary_input(temp_step->primary_input());
               a_step->secondary_input(temp_step-
>secondary_input());
               a_step->affected_module(temp_step-
>affected_module());
               a_step->set_priority(temp_step-
>get_priority());
              a_step->preceded_by(temp_step->preceded_by());
               a_step->set_estimated_duration(temp_step-
```

```
>get_estimated_duration());
                a_step->set_deadline(temp_step-
>get_deadline());
                a_step-
>set_required_expertise_level(temp_step-
>get_required_expertise_level());
                temp_step->deleteObject();
              a_step->putObject();
            }
          }
        OC_transactionCommit();
        OC_close();
}
void create_step(char* dbname, char* pr_Name, char*
comp_name)
{
  OC_open(dbname);
  OC_transactionStart();
  int my_step_id=0;
  int var, ver, var1,ver1;
  char protoName[64];
  char* my_step_name="step_";
  char theName[64];
  char temp[64];
  Set *aSet = (Set *)OC_lookup("step_set");
    // If it does not exist, create it
  if(aSet == NULL) {
      //cout << "Creating set object ...\n";

      // Create a new set called step_set
      aSet = new Set((Type*)OC_lookup("STEP"), "step_set");
      }
  // Create step objects and insert it into the step_set
   Sequencer*
aSequencer=(Sequencer*)OC_lookup("Step_Sequencer");
   if(aSequencer== NULL){
       //cout << "Creating Sequencer  object ...\n";
       aSequencer= new Sequencer("Step_Sequencer");
//       aSequencer->putObject();
       my_step_id=1;
       }
    else{
//       cout << aSequencer->Name() << " already exists.\n";
       my_step_id = aSequencer->getValue();
//       aSequencer->putObject();
       }
```

```
    sprintf(temp,"%s%d", my_step_name,my_step_id);
    STEP* a_step=new STEP(temp,my_step_id);

    sscanf(pr_Name,"%s %d:%d",theName, &var, &ver);
    char *aName = new char[strlen(theName)+1];
    strcpy(aName,theName);
    if(var==0) var1 = 1;
    else var1 =var;
    if(ver==0) ver1 =1;
    else ver1=ver;
    sprintf(protoName,"%s %d:%d",aName, var1, ver1);
    thepath=0;
    find_component_path(protoName,aName);
    if(!thepath){
       cout << "there is no: "<<aName <<"   prototype in DDB
\n";
       return;
      }
   else{
       COMP_REFERENCE* my_ref=new COMP_REFERENCE();
       my_ref->set_priv_name(aName);
       my_ref->set_version_no(ver);
       my_ref->set_variation_no(var);
       my_ref->putObject();
       a_step->set_base_version(my_ref);
          sscanf(comp_name,"%s %d:%d",theName, &var, &ver);
            char* a_name=new char[strlen(theName)+1];
          strcpy(a_name,theName);
          char* check=0;
          check = theName+strlen(theName)-10;
          if(strcmp(check,".spec.psdl")==0){
               My_String temp1= My_String(theName)-10;
             char* check1=(char*)temp1;
             find_component_path(protoName,check1);
             if(!thepath)
             cout <<"there is no: "<<theName <<"   component
in DDB \n";
             else{
               COMP_REFERENCE* my_ref=new COMP_REFERENCE();
               my_ref->set_priv_name(a_name);
               my_ref->set_version_no(ver);
               my_ref->set_variation_no(var);
               my_ref->putObject();
             set_affected_modules(a_step, protoName,check1);
               a_step->add_primary_input(my_ref);
               aSequencer->putObject();
```

```
                a_step->putObject();
                aSet->Insert(a_step);
                aSet->putObject();
                }
            }else{
               char* check1 =0;
                 check1 = theName+strlen(theName)-9;
               if(strcmp(check1,".imp.psdl")==0){
               My_String temp1= My_String(theName)-9;
               char* check1=(char*)temp1;
                  find_component_path(protoName,check1);
                  if(!thepath)
                cout <<"there is no: "<<theName <<"   component
in DDB \n";
                  else{
                  COMP_REFERENCE* my_ref=new COMP_REFERENCE();
                     my_ref->set_priv_name(a_name);
                     my_ref->set_version_no(ver);
                     my_ref->set_variation_no(var);
                     my_ref->putObject();
                     a_step->add_primary_input(my_ref);
                     aSequencer->putObject();
                         set_secondary_input(a_step,thepath);
                     strcat(check1,".spec.psdl");
                     my_ref=new COMP_REFERENCE();
                     my_ref->set_priv_name(check1);
                     my_ref->putObject();
                     a_step->add_secondary_input(my_ref);
                     a_step->putObject();
                     aSet->Insert(a_step);
                     aSet->putObject();
                       }
                   }else
                      cout << "wrong components suffix \n";

               }
         }
     OC_transactionCommit();
     OC_close();
}

void set_affected_modules(STEP* a_step, char* protoName,char*
comp_name)
{
           char* aName= new char[strlen(comp_name)+1];
         strcpy(aName,comp_name);
```

```
          COMP_REFERENCE* my_ref=new COMP_REFERENCE();
          char temp[64];
          sscanf(protoName,"%s", temp);
          if (strcmp(temp,comp_name)!=0){
               char *separator=".";
             char* word[5];
             int i=0;
             word[0]=strtok(comp_name, separator);
             while(word[i] !=NULL){
                  i=i+1;
                  word[i]=strtok(NULL, separator);
                     }
          int n = strlen(word[i-1])+1;
          My_String temp=My_String(aName)- n
+My_String(".imp.psdl");
          char* parent=(char*)temp;
             my_ref->set_priv_name(parent);
             my_ref->putObject();
             a_step->add_affected_modules(my_ref);
             }
        strcat(aName,".imp.psdl");
        my_ref=new COMP_REFERENCE();
          my_ref->set_priv_name(aName);
          my_ref->putObject();
          a_step->add_affected_modules(my_ref);
        a_step->putObject();
}

void set_secondary_input(STEP* a_step,char* compPath)
{
          COMP_REFERENCE* my_ref;
          char * check=(char*)0;
       COMPONENT *my_comp =(COMPONENT *)OC_lookup(compPath);
        if(my_comp != NULL){
        List* my_list =  my_comp->subComponents();
          ListIterator my_iterator(my_list);
        COMPONENT *theComponent;
        while(my_iterator.moreData())
         {
          theComponent=(COMPONENT *)(Entity*)my_iterator();
            List* the_list = theComponent->TextObjectList();
          ListIterator the_iterator(the_list);
            TEXT_OBJECT *the_text_object;
              while(the_iterator.moreData())
              {
              the_text_object =
```

```
(TEXT_OBJECT*)(Entity*)the_iterator();
              check =the_text_object->getFileName()+
                  strlen(the_text_object->getFileName())-10;
            if(strcmp(check,".spec.psdl")==0){
            my_ref=new COMP_REFERENCE();
            my_ref->set_priv_name(the_text_object-
>getFileName());
              my_ref->putObject();
            a_step->add_secondary_input(my_ref);
              }
            }
            }
        }
      else
        cout <<"cannot find the component \n";
}


void create_substep(char* dbname,int step_id, char* p_input,
                        char* d_name, char* thedate, int
duration)
{
  OC_open(dbname);
  OC_transactionStart();
  OC_Boolean FOUND=FALSE;
  Time T1(thedate);
  int my_step_id=0;
  char temp[64];
  char temp1[64];
  char* my_step_name="step_";
  Set *aSet = (Set *)OC_lookup("step_set");
  Sequencer*
aSequencer=(Sequencer*)OC_lookup("Step_Sequencer");
  my_step_id = aSequencer->getValue();
  aSequencer->putObject();
  sprintf(temp,"%s%d", my_step_name,step_id);
  STEP* the_step=(STEP*)OC_lookup (temp);
  if (the_step == NULL)
    cout <<"there is no such parent step \n";
  else{
    sprintf(temp1,"%s%d", my_step_name,my_step_id);
    STEP* a_step=new STEP(temp1,my_step_id);
    if(p_input[0]!= '0'){
    COMP_REFERENCE* my_ref=new COMP_REFERENCE();
    my_ref->set_priv_name(p_input);
    my_ref->putObject();
```

```
    a_step->add_primary_input(my_ref);
    a_step->set_parent_step(the_step);
    a_step->set_base_version(the_step->get_base_version());
    a_step->set_deadline(the_step->get_deadline());
    a_step->set_priority(the_step->get_priority());
    a_step->set_estimated_duration(duration);
    a_step->set_step_type(5);
    List* my_list= the_step->substep();
    ListIterator my_Iterator(my_list);
  STEP* my_step;
  // For each item in the iterator
  while(my_Iterator.moreData()&& !FOUND) {
      // Get the item
        my_step=(STEP*)(Entity *)my_Iterator() ;
        if(my_step->get_status()==3 &&
           strcmp(my_step->get_designer(),d_name)==0){
           a_step->add_predecessor(my_step);
           FOUND = TRUE;
        }
    }
    a_step->putObject();

           cout << a_step->get_step_id()<< "\n";
           int T2= a_step->get_deadline() - T1;
           if (T2 < 0)
           cout <<"1000" << "\n";
           else
           cout <<T2 << "\n";
           cout <<a_step->get_priority() << "\n";
           cout <<a_step->get_estimated_duration() <<"\n";
           cout <<"{";
           a_step->show_preceding_steps();
           cout <<"} \n";
           cout <<Level[a_step-
>get_required_expertise_level()]<< "\n";
           cout << a_step->get_in_degree() << "\n";

    the_step->add_substep(a_step);
    char protoname[64];
    My_String temp4= My_String(p_input)-9;
    char *compname = (char*) temp4;
    COMP_REFERENCE* my_comp= the_step->get_base_version();
    char *temp2=my_comp->get_priv_name();
    int var=my_comp->get_variation_no();
    int ver=my_comp->get_version_no();
    sprintf(protoname,"%s %d:%d", temp2,var, ver);
```

```
    find_component_path(protoname, compname);
    if (thepath)
        set_secondary_input(a_step,thepath);
//    else
//       cout <<"Cannot find component: " << compname <<" in
DDB \n";
    }
    the_step->putObject();
    aSet->Insert(a_step);
    aSet->putObject();
    }
    OC_transactionCommit();
    OC_close();
}


STEP* create_substep(STEP* the_step, char* p_input)
{
//  OC_open(dbname);
//  OC_transactionStart();
  int my_step_id=0;
  char temp1[64];
  char* my_step_name="step_";
  Set *aSet = (Set *)OC_lookup("step_set");
  Sequencer*
aSequencer=(Sequencer*)OC_lookup("Step_Sequencer");
  my_step_id = aSequencer->getValue();
  aSequencer->putObject();
    sprintf(temp1,"%s%d", my_step_name,my_step_id);
    STEP* a_step=new STEP(temp1,my_step_id);
//    cout <<"step name: " <<temp1 <<"\n";
    COMP_REFERENCE* my_ref=new COMP_REFERENCE();
    char* a_name=0;
    a_name=new char[strlen(p_input)+1];
    strcpy(a_name,p_input);
    my_ref->set_priv_name(a_name);
    my_ref->putObject();
    a_step->set_base_version(the_step->get_base_version());
    a_step->add_primary_input(my_ref);
    a_step->set_parent_step(the_step);
    a_step->putObject();
    the_step->add_substep(a_step);
    char protoname[64];
    My_String temp4= My_String(p_input)-9;
    char *compname = (char*) temp4;
    COMP_REFERENCE* my_comp= the_step->get_base_version();
```

```
    char *temp2=my_comp->get_priv_name();
    int var=my_comp->get_variation_no();
    int ver=my_comp->get_version_no();
    sprintf(protoname,"%s %d:%d", temp2,var, ver);
    char* check=a_name+strlen(a_name)-10;
    if(strcmp(check,".spec.psdl") != 0)
    {
    thepath =0;
    find_component_path(protoname, compname);
    if (thepath){
//        cout <<"check: " <<check <<"\n";
        set_secondary_input(a_step,thepath);
        strcat(compname,".spec.psdl");
        COMP_REFERENCE* my_ref=new COMP_REFERENCE();
        my_ref->set_priv_name(compname);
        my_ref->putObject();
        a_step->add_secondary_input(my_ref);
          }
    else
        cout <<"Cannot find component: " << compname <<" in
DDB \n";
    }
    a_step->putObject();
    the_step->putObject();
    aSet->Insert(a_step);
    aSet->putObject();
    return a_step;
}

void auto_create_substeps(char* dbname,int step_id)
{
  OC_open(dbname);
  OC_transactionStart();
  COMP_REFERENCE* my_ref;
  STEP* my_step;
  STEP* a_step1;
  STEP* a_step2;
  char temp[64];
  char* my_step_name="step_";
  sprintf(temp,"%s%d", my_step_name,step_id);
  STEP* the_step=(STEP*)OC_lookup (temp);
  if (the_step == NULL)
    cout <<"there is no such parent step \n";
  else{
    List *my_list=the_step->primary_input();
    my_ref =(COMP_REFERENCE*)my_list->getEntityElement(0);
```

```
//    char*  my_name=0;
    int n1= strlen(my_ref->get_priv_name())+1;
    char* my_name=new char[n1];
    strcpy(my_name,my_ref->get_priv_name());
    my_name[n1]='\0';
    my_step= create_substep(the_step,my_name);
//    cout <<"my_name: " <<my_name <<"\n";
      char* check=0;
      check = my_ref->get_priv_name()+strlen(my_ref-
>get_priv_name())-10;
//   cout<<"check: " << check <<"\n";
      if(strcmp(check,".spec.psdl")==0){
         my_step->set_step_type(1);
         my_step->set_status(1);
         my_step->set_deadline(the_step->get_deadline());
         my_step->set_priority(the_step->get_priority());
         the_step->set_status(1);
         the_step->putObject();
         my_step->putObject();
         List* a_list = the_step->affected_module();
         COMP_REFERENCE* a_ref1;
         COMP_REFERENCE* a_ref2;
           a_ref1 =(COMP_REFERENCE*)a_list-
>getEntityElement(0);
//         char* a_name2=0;
         int n2=strlen(a_ref1->get_priv_name())+1;
         char* a_name2= new char[n2];
         strcpy(a_name2,a_ref1->get_priv_name());
         a_name2[n2]='\0';
//         cout <<"my_name: " <<a_name2 <<"\n";
           a_step1 = create_substep(the_step,a_name2);
         if(a_list->Cardinality()==1)
            a_step1->set_step_type(2);
         else
              a_step1->set_step_type(3);
         a_step1->set_status(1);
           a_step1->add_predecessor(my_step);
         a_step1->set_deadline(the_step->get_deadline());
         a_step1->set_priority(the_step->get_priority());
           a_step1->putObject();
         if(a_list->Cardinality()> 1){
           a_ref2 =(COMP_REFERENCE*)a_list-
>getEntityElement(1);
         int n3=strlen(a_ref2->get_priv_name())+1;
           char* a_name1= new char[n3];
//         cout <<"my_name: " <<a_name1 <<"\n";
```

```
          strcpy(a_name1,a_ref2->get_priv_name());
          a_name1[n3]='\0';
            a_step2 = create_substep(the_step,a_name1);
            a_step2->set_step_type(2);
          a_step2->set_status(1);
            a_step2->add_predecessor(my_step);
          a_step2->set_deadline(the_step->get_deadline());
          a_step2->set_priority(the_step->get_priority());
            a_step2->putObject();
          my_step->putObject();
          }
            }
        else{
          my_step->set_step_type(4);
          my_step->secondary_input(the_step-
>secondary_input());
          my_step->set_status(1);
          my_step->set_deadline(the_step->get_deadline());
          my_step->set_priority(the_step->get_priority());
          my_step->putObject();
          the_step->set_status(1);
          the_step->putObject();
        }
    }
     OC_transactionCommit();
     OC_close();
}

void commit_substep(char* dbname,int step_id)
{
  OC_open(dbname);
  OC_transactionStart();
  STEP* the_step=find_step(step_id);
  if (the_step == NULL)
    cout <<"there is no such step \n";
  else{
    int my_status = the_step->get_status();
    if( my_status !=3)
    cout << "cannot commit an un-assigned step \n";
    else{
        if(the_step->get_step_type()==0)
          cout <<the_step->Name() << " is not a substep!! \n";
        else{
    char an_id[64];
    sprintf(an_id,"%d",the_step->get_step_id());
    My_String
```

```
temp=My_String(".")+My_String("step_")+My_String(an_id);
    dirNamePtr =(char*) temp;

              COMP_REFERENCE* my_ref= the_step-
>get_base_version();
            char* thename = my_ref->get_priv_name();
            int var = my_ref->get_variation_no();
            int ver = my_ref->get_version_no();
            char protoname[64];
            sprintf(protoname, "%s %d:%d", thename,var,ver);
//            cout <<"protoname: " <<protoname <<"\n";
              List* my_list= the_step->primary_input();
              COMP_REFERENCE* a_ref1;
            a_ref1 =(COMP_REFERENCE*)my_list-
>getEntityElement(0);
            char *my_name= new char[strlen(a_ref1-
>get_priv_name())+1];
            strcpy(my_name,a_ref1->get_priv_name());
              if(the_step->get_step_type()==1){
               My_String temp4= My_String(my_name)-10;
               char *compName = (char*) temp4;
//               cout << "compName: " <<compName <<"\n";

               COMPONENT* new_comp;
              new_comp = Add_new_version(protoname,compName);
               if(new_comp== NULL)
                  cout << "the dam error again \n";
               else{
                    the_step->add_output(new_comp);
                    the_step->set_status(4);
                    the_step->putObject();
                        }
                 }
            else{
               if(the_step->get_step_type()== 4){
               My_String temp4= My_String(my_name)-9;
               char *compName = (char*) temp4;
               COMPONENT* new_comp;
              new_comp = Add_new_version(protoname,compName);
               if(new_comp== NULL)
                  cout << "the dam error again \n";
               else{
                    the_step->add_output(new_comp);
                    the_step->set_status(4);
                    the_step->putObject();
                    }
```

```
            }
           else
            {
            My_String temp4= My_String(my_name)-9;
            char *compName = (char*) temp4;
            List* a_list= the_step->preceded_by();
            STEP* my_step=(STEP*)a_list->getEntityElement(0);
            List* the_list=my_step->output();
            COMPONENT* aComp=(COMPONENT*)
                  the_list->getEntityElement(0);
            if(the_step->get_step_type()==2){
                   My_String temp1(My_String(dirNamePtr)
          +My_String("/")+ My_String(a_ref1-
>get_priv_name()));
              char *temp2= (char*) temp1;
              ifstream myFile(temp2); //=(char*)0;
              if(myFile){
                 TEXT_OBJECT *a_text_obj;
                 cout <<"Creating text object: "
                     <<a_ref1->get_priv_name() <<"...\n";
                 a_text_obj= new TEXT_OBJECT();
            a_text_obj->append(a_ref1->get_priv_name(),
                                    myFile);
                 aComp->replaceTextObject(a_text_obj);
                 aComp->putObject();
                 the_step->add_output(aComp);
                 the_step->set_status(4);
                 the_step->putObject();
                     }
           else cout <<"there is no such file: " <<a_ref1-
>get_priv_name() <<"\n";
           }
             else{
               if(the_step->get_step_type()==3){
               char* badr= new char[strlen(aComp-
>CompnentName())+1];
               strcpy(badr,aComp->CompnentName());
               char* salah=get_last_token(badr);
                 COMPONENT* new_comp;
                 new_comp =
Add_new_version(protoname,compName);
                 if(new_comp != NULL){
            new_comp->replace_subconponent(aComp,salah);
                 new_comp->putObject();
                 the_step->add_output(new_comp);
                 the_step->set_status(4);
```

```
                    the_step->putObject();
                    }
                    }
                  }
                }
              }
            }
          }
        }
    OC_transactionCommit(OC_doNothing);
    OC_close();
}

void commit_step(char* dbname,int step_id)
{
  OC_open(dbname);
  OC_transactionStart();
  char temp[64];
  char* my_step_name="step_";
  sprintf(temp,"%s%d", my_step_name,step_id);
  STEP* the_step=(STEP*)OC_lookup (temp);
  if (the_step == NULL)
    cout <<"there is no such step \n";
  else{
       int my_status = the_step->get_status();
       if( my_status !=3)
       cout << "cannot commit un-assigned step \n";
       else{
           COMP_REFERENCE* my_ref= the_step-
>get_base_version();
           char* thename = my_ref->get_priv_name();
           int var = my_ref->get_variation_no();
           int ver = my_ref->get_version_no();
           char protoname[64];
           sprintf(protoname, "%s %d:%d", thename,var,ver);

           if(the_step->get_step_type() ==0)
             {
               List* my_list = the_step->substep();
               if(my_list->Cardinality()== 1)
                   {
                   STEP* my_step=(STEP*)my_list-
>getEntityElement(0);
               List* a_list = my_step->output();
               COMPONENT* my_comp;
               my_comp =(COMPONENT*)a_list-
```

```
>getEntityElement(0);
                char*
temp1=generate_new_configuration(protoname, my_comp);
                update_base_versions(step_id,temp1);
                the_step->set_status(4);
                the_step->putObject();
                    }
              else
                {
                    List* my_list = the_step->substep();
                ListIterator my_iterator(my_list);
                while(my_iterator.moreData())
                   {
                        STEP*
my_step=(STEP*)(Entity*)my_iterator();
                        if(my_step->get_step_type()==3)
                          {
                       List* a_list = my_step->output();
                          COMPONENT* my_comp;
                          my_comp =(COMPONENT*)a_list-
>getEntityElement(0);
                          char*
temp1=generate_new_configuration(protoname, my_comp);
                      the_step->set_status(4);
                      update_base_versions(step_id,temp1);
                          the_step->putObject();
                          }
                        }
                    }
                   }
                else
              cout <<" Sorry this is not a top level step \n";
              }
           }
    OC_transactionCommit(OC_doNothing);
    OC_close();
}


void update_base_versions(int step_id,char* temp1)
{
   char propto_name[64];
   int var, ver;
   sscanf(temp1,"%s %d:%d", propto_name, &var, &ver);
   STEP* the_step = find_step(step_id);
   if(the_step != NULL){
```

```
        COMP_REFERENCE* a_ref= the_step->get_base_version();

         Set *aSet = (Set *)OC_lookup("step_set");
         // Ask the set object for an iterator
         Iterator* anIterator = aSet->getIterator();
         STEP* my_step;
         // For each item in the iterator
         while(anIterator->moreData()) {
         // Get the item
         my_step=(STEP*)(Entity *)(anIterator->operator()());
         if(my_step->get_status() <= 3){
         COMP_REFERENCE* my_ref= my_step->get_base_version();
         if(strcmp(a_ref->get_priv_name(),
                     my_ref->get_priv_name())==0 &&
          a_ref->get_version_no()== my_ref->get_version_no()
&&
             a_ref->get_variation_no()== my_ref-
>get_variation_no()){
            my_ref->set_version_no( ver);
            my_ref->set_variation_no(var);
            my_ref->putObject();
            my_step->putObject();
            }
           }
          }
      }
}
```

mainstep.cxx **********

```
#include <Database.h>
#include <Directory.h>
#include <string.h>
#include <Set.h>
#include "step_Operations.h"

extern "C"
{
          char *getenv(const char *);
}

//Globals
char* dirNamePtr=".";
int warning_time=1;
char *dbName= "supportDB";
char* DESIGN_DATABASE_DIRECTORY="DesignDB";
```

```
char* thepath =(char*)0;
char* v_path =(char*)0;
COMPONENT* compPtr=NULL;

int main(int argc,char *argv[])
{
        char *option=(char*)0;
        char *aName=(char*)0;
        char *aName1=(char*)0;
        char *dbName=(char*)0;
        char theothername[64];
        char tmp1[64];
        char tmp2[64];
        char *tmp3=(char*)0;
        char tmp4[64];
        char *tmp5=(char*)0;
        char tmp6[64];
        char *tmp7=(char*)0;
        int my_step_id;
        int a_step_id, a_value,a_value1,a_value2;

        char *user_name = getenv("USER");

        if (argv[1])
        {
                dbName = new char[strlen(argv[1])+1];
                strcpy(dbName,argv[1]);
        }

        if (argv[2])
        {
                option = new char[strlen(argv[2])+1];
                strcpy(option, argv[2]);
        }

        // Create step
            if(option[0] =='1'){
                // get proto name
                    aName1 = new char[strlen(argv[3])+1];
                strcpy(aName1,argv[3]);
                aName = new char[strlen(argv[4])+1];
                strcpy(aName,argv[4]);
                sprintf(tmp1,"%s %s",aName1,aName);
                aName1 = new char[strlen(argv[5])+1];
                strcpy(aName1,argv[5]);
                aName = new char[strlen(argv[6])+1];
```

```
            strcpy(aName,argv[6]);
            sprintf(tmp2,"%s %s",aName1,aName);
            create_step(dbName,tmp1,tmp2);

    }
    else if(option[0] == '2'){
            //Show Step
                aName = new char[strlen(argv[3])+1];
            strcpy(aName,argv[3]);
            sscanf(aName,"%d",&my_step_id);
            show_step(dbName,my_step_id);
    }
    else if(option[0] == '3'){
            // show steps of certain status
                aName = new char[strlen(argv[3])+1];
            strcpy(aName,argv[3]);
            show_steps(dbName,aName);
    }
    else if(option[0] == '4'){
            // update step
                aName = new char[strlen(argv[3])+1];
            strcpy(aName,argv[3]);
            sscanf(aName,"%d",&my_step_id);

            // update deadline
                aName1 = new char[strlen(argv[4])+1];
            strcpy(aName1,argv[4]);
            aName = new char[strlen(argv[5])+1];
            strcpy(aName,argv[5]);
            sprintf(tmp1,"%s %s",aName1,aName);

            // add primary input
                aName1 = new char[strlen(argv[6])+1];
            strcpy(aName1,argv[6]);
            aName = new char[strlen(argv[7])+1];
            strcpy(aName,argv[7]);
            sprintf(tmp2,"%s %s",aName1,aName);

            //delete primary input
                tmp3 = new char[strlen(argv[8])+1];
            strcpy(tmp3,argv[8]);

            // add secondary input
                aName1 = new char[strlen(argv[9])+1];
            strcpy(aName1,argv[9]);
            aName = new char[strlen(argv[10])+1];
```

```
                    strcpy(aName,argv[10]);
                    sprintf(tmp4,"%s %s",aName1,aName);

                    // delete secondary input
                        tmp5 = new char[strlen(argv[11])+1];
                    strcpy(tmp5,argv[11]);

                    // add affected_modules
                        aName1 = new
char[strlen(argv[12])+1];
                    strcpy(aName1,argv[12]);
                    aName = new char[strlen(argv[13])+1];
                    strcpy(aName,argv[13]);
                    sprintf(tmp6,"%s %s",aName1,aName);

                    // delete affected_modules
                        tmp7 = new char[strlen(argv[14])+1];
                    strcpy(tmp7,argv[14]);

                    // Update_priority
                        aName1 = new
char[strlen(argv[15])+1];
                    strcpy(aName1,argv[15]);
                    sscanf(aName1,"%d",&a_value);

                    // Update_precedence
                        aName1 = new
char[strlen(argv[16])+1];
                    strcpy(aName1,argv[16]);
                    sscanf(aName1,"%d",&a_step_id);

                    //Update_estimated_duration
                        aName1 = new
char[strlen(argv[17])+1];
                    strcpy(aName1,argv[17]);
                    sscanf(aName1,"%d",&a_value1);

                    // Update_expertise_level
                        aName1 = new
char[strlen(argv[18])+1];
                    strcpy(aName1,argv[18]);
                    sscanf(aName1,"%d",&a_value2);
                    Update_Step(dbName,my_step_id,tmp1,
                             tmp2,tmp3,tmp4,tmp5,tmp6,tmp7,

a_value,a_step_id,a_value1,a_value2);
```

```
        }
        else if(option[0] == '5'){
                // Update_start_time
                    aName = new char[strlen(argv[3])+1];
                strcpy(aName,argv[3]);
                sscanf(aName,"%d",&my_step_id);
                aName1 = new char[strlen(argv[4])+1];
                strcpy(aName1,argv[4]);
                aName = new char[strlen(argv[5])+1];
                strcpy(aName,argv[5]);
                sprintf(theothername,"%s
%s",aName1,aName);
                Update_start_time(dbName,my_step_id,

theothername);
        }
        else if(option[0] == '6'){
                // Update_finish_time
                    aName = new char[strlen(argv[3])+1];
                strcpy(aName,argv[3]);
                sscanf(aName,"%d",&my_step_id);
                aName1 = new char[strlen(argv[4])+1];
                strcpy(aName1,argv[4]);
                aName = new char[strlen(argv[5])+1];
                strcpy(aName,argv[5]);
                sprintf(theothername,"%s
%s",aName1,aName);
                Update_finish_time(dbName,my_step_id,

theothername);
        }
        else if(option[0] == '7'){
                // Update Status
                    aName = new char[strlen(argv[3])+1];
                strcpy(aName,argv[3]);
                sscanf(aName,"%d",&my_step_id);
                aName1 = new char[strlen(argv[4])+1];
                strcpy(aName1,argv[4]);
                sscanf(aName1,"%d",&a_value);

Update_status(dbName,my_step_id,a_value);
        }
        else if(option[0] == '8'){
                // Update designer
                    aName = new char[strlen(argv[3])+1];
                strcpy(aName,argv[3]);
```

```
                    sscanf(aName,"%d",&my_step_id);
                    aName1 = new char[strlen(argv[4])+1];
                    strcpy(aName1,argv[4]);

Update_designer(dbName,my_step_id,aName1);
          }
          else if(option[0] == '9'){
                    // print commit data
                        aName = new char[strlen(argv[3])+1];
                    strcpy(aName,argv[3]);
                    sscanf(aName,"%d",&my_step_id);
                    get_commit_data(dbName,my_step_id);
          }
          else if(option[0] == 'a'){
                    // get scheduling data
                        aName = new char[strlen(argv[3])+1];
                    strcpy(aName,argv[3]);
                    sscanf(aName,"%d",&my_step_id);
                    aName1 = new char[strlen(argv[4])+1];
                    strcpy(aName1,argv[4]);
                    aName = new char[strlen(argv[5])+1];
                    strcpy(aName,argv[5]);
                    sprintf(theothername,"%s
%s",aName1,aName);
                    get_scheduling_data(dbName,my_step_id,

theothername);
          }
          else if(option[0] == 'b'){
                    // create substep
                    aName = new char[strlen(argv[3])+1];
                    strcpy(aName,argv[3]);
                    sscanf(aName,"%d",&my_step_id);
                    aName1 = new char[strlen(argv[4])+1];
                    strcpy(aName1,argv[4]);
                    tmp3 = new char[strlen(argv[5])+1];
                    strcpy(tmp3,argv[5]);
                    tmp5 = new char[strlen(argv[6])+1];
                    strcpy(tmp5,argv[6]);
                    sprintf(theothername,"%s %s",tmp3,tmp5);
                    aName = new char[strlen(argv[7])+1];
                    strcpy(aName,argv[7]);
                    sscanf(aName,"%d",&a_value);
                    create_substep(dbName,my_step_id,aName1,
                              user_name, theothername,
a_value);
```

```
        }
        else if(option[0] == 'c'){
                    // approve step
                        aName = new char[strlen(argv[3])+1];
                    strcpy(aName,argv[3]);
                    sscanf(aName,"%d",&my_step_id);
                    auto_create_substeps(dbName,my_step_id);
        }
        else if(option[0] == 'd'){
                    // commit substep
                        aName = new char[strlen(argv[3])+1];
                    strcpy(aName,argv[3]);
                    sscanf(aName,"%d",&my_step_id);
                    commit_substep(dbName,my_step_id);
        }
        else if(option[0] == 'e'){
                    // commit step
                        aName = new char[strlen(argv[3])+1];
                    strcpy(aName,argv[3]);
                    sscanf(aName,"%d",&my_step_id);
                    commit_step(dbName,my_step_id);
        }
        else if(option[0] == 'f'){
                    // remove_step_from_schedule
                        aName = new char[strlen(argv[3])+1];
                    strcpy(aName,argv[3]);
                    aName1 = new char[strlen(argv[4])+1];
                    strcpy(aName1,argv[4]);
                    tmp3 = new char[strlen(argv[5])+1];
                    strcpy(tmp3,argv[5]);
                    sprintf(theothername,"%s
%s",aName1,tmp3);
                    remove_step_from_schedule(dbName,aName,

theothername);
        }
        else if(option[0] == 'g'){
                    // Dump step components
                        aName = new char[strlen(argv[3])+1];
                    strcpy(aName,argv[3]);
                    sscanf(aName,"%d",&my_step_id);
                    dump_step_components(dbName,my_step_id);
        }
        else if(option[0] == 'h')
                    // find assigned step
                        find_assigned_step(dbName,
```

```
user_name);
          else if(option[0] == 'i'){
                    // Suspend/Abandon Step
                        aName = new char[strlen(argv[3])+1];
                    strcpy(aName,argv[3]);
                    sscanf(aName,"%d",&my_step_id);
                    aName1 = new char[strlen(argv[4])+1];
                    strcpy(aName1,argv[4]);
                    sscanf(aName1,"%d",&a_value);
                    suspend_abandon_step(dbName,my_step_id,
                                                    a_value);
          }
          else if(option[0] == 'j'){
                    // Suspend/Abandon Step
                        aName = new char[strlen(argv[3])+1];
                    strcpy(aName,argv[3]);
                    sscanf(aName,"%d",&my_step_id);
                    Update_deadline(dbName,my_step_id);
          }
          else if(option[0] == 'k'){
                    // Suspend/Abandon Step
                        aName = new char[strlen(argv[3])+1];
                    strcpy(aName,argv[3]);
                    aName1 = new char[strlen(argv[4])+1];
                    strcpy(aName1,argv[4]);
                    sprintf(theothername,"%s %s",aName,
aName1);
                    Early_warning(dbName,theothername);
          }
          else if(option[0] == 'l'){
                    // get scheduling data
                        aName = new char[strlen(argv[3])+1];
                    strcpy(aName,argv[3]);
                    aName1 = new char[strlen(argv[4])+1];
                    strcpy(aName1,argv[4]);
                    sprintf(theothername,"%s
%s",aName,aName1);
                    aName = new char[strlen(argv[5])+1];
                    strcpy(aName,argv[5]);

get_scheduling_data_2(dbName,theothername,

aName);
          }
          else if(option[0] == 'm'){
                    aName = new char[strlen(argv[3])+1];
```

```
                    strcpy(aName,argv[3]);
                    aName1= new char[strlen(argv[4])+1];
                    strcpy(aName1,argv[4]);
                    sprintf(theothername,"%s
%s",aName,aName1);
                    aName = new char[strlen(argv[5])+1];
                    strcpy(aName,argv[5]);
                    get_Sched_data_1(dbName,"MySchedule" ,
                                        theothername,aName);
          }
          else
                    cout <<"Wrong Option: " << option << "
Try again \n";
}
```

2. Class Component

component.h **********

```
#ifndef __COMPONENT_H
#define __COMPONENT_H

#include <Object.h>
#include <Dictionary.h>
#include <Type.h>
#include <List.h>
#include <Reference.h>
//#include "ReferenceMacros.h"
#include <stream.h>

#ifndef __TEXT_OBJECT_H
#include "text_object.h"
#endif


extern "C"
{
#include <sys/time.h>
#include <sys/types.h>
#include <string.h>
}

// Class //
//    Defines a COMPONENT class.  The class COMPONENT is a
derived
```

```
//    class of Object.
class COMPONENT :
public Object
{
private :
          int                    version_id;
          int                    variation_id;
          char*                  theName;
          char*                  priv_author;
          time_t                 DateCreated;
          char*                  previous_version;
          char*                  next_version;
          Reference              part_of_list;
          // parent relationship
          Reference              sub_component_list;
          // child relationship
          Reference              used_by_list;
          Reference              text_object_list;

public:

          //    COMPONENT --APL ONTOS required constructor
              COMPONENT(APL *theAPL);

          //    Constructs a COMPONENT object
              COMPONENT(char* name,int ver, int var);

          // ONTOS method for savig Object as pesistent
Object.
              virtual void putObject(OC_Boolean
deallocate=FALSE);

          // ONTOS method for deleting an Object
              virtual void deleteObject(OC_Boolean
                                        deallocate=FALSE);

          // ONTOS heap mangagement method.
              virtual void Destroy(Boolean abort = FALSE);

          //    Return the ONTOS Type of class COMPONENT.
              virtual Type *getDirectType();

          // Set the version mumber
              void versionNumber(int ver){
                    version_id = ver;
              }
```

```
// get the version mumber
    int versionNumber(){
          return version_id;
}

// Set the variation mumber
    void variationNumber(int var){
          variation_id = var;
}

// get the variation mumber
    int variationNumber(){
          return variation_id;
}

// Set the author name
    void AuthorName(char* auth){
          priv_author = auth;
}

// get the author name
    char* AuthorName(){
          return priv_author;
}

// Set the component name
    void CompnentName(char* a_name){
          theName = a_name;
}

// get the component  name
    char* CompnentName(){
          return theName;
}

// set creation time
    time_t setCreationDate();

// get creation time
    time_t getCreationDate(){
          return DateCreated;
}

// set previous_version
    void setPrevious_version(char* comppath)
```

```
            {
                  previous_version = comppath;
            }

            // get previous_version
                char* getPrevious_version(){
                  return previous_version;
            }

            // set next_version
                void setNext_version(char* comppath)
                {
                  next_version = comppath;
            }

            char* getNext_version(){
                  return next_version;
            }

            // add a subcomponent
              void addSubcomponent(COMPONENT* otherComponent);

            // add a parent to the parent list
            void   addParentComponent(COMPONENT*
otherComponent);
            void replace_subconponent(COMPONENT* my_comp, char

*comp_name);
            // find component
                void find_component(char *thename);

            // search the tree for a component
                void find_a_component(char *thename);

            //search the tree for a parent of a component
                void find_parent(char *thename);

            // get a list of subcomponents
                List* subComponents(){
                  return (List*)
                  sub_component_list.Binding(this);
            }

            // reset a list of subcomponents
                void  subComponents( List* parts) {
                  sub_component_list.Reset(parts, this);
```

```
        }

        // get a list of text objects
            List* TextObjectList(){
                  return (List*)
                  text_object_list.Binding(this);
        }

        // reset a list of text objects
            void  TextObjectList( List* parts) {
                  text_object_list.Reset(parts, this);
        }

        // delete the text objects of a COMPONENT object.
            void deleteComponentText();

        // Display the file names of the files contained in
        // the COMPONENT node
            void getComponentNames();

        // Display the file names of the file contained in
        // each COMPONENT of the subtree
            void getComponentSubtreeNames();

        //Output the contents of an COMPONENT node to files.
            Boolean getComponentSource(char*);
            void getComponentSubtreeSource(char*);
        // Inserts a text_object into the COMPONENT node.
            void addTextObject(TEXT_OBJECT *);

        void replaceTextObject(TEXT_OBJECT
*my_text_object);
        //  Output the .ps,.spec,..and .a  files contained i
        // in the COMPONENT node.
        Boolean getPSfile(char*);

        Boolean getGRAPHfile(char*);

        Boolean getSPECfile(char*);

        Boolean getIMPfile(char*);

        Boolean getSOURCEfile(char*);

        char  *getTEXTPtr(char*);
```

```
        //    Destructor for the COMPONENT class.
            ~COMPONENT() {
                  Destroy(FALSE);
            };
};

class Sequencer : public Object
{
private :
            int    theValue;
public:
            Sequencer(APL *theAPL);
            Sequencer(char* a_name);
            Type *getDirectType();

            //increment the sequencer and return the new value.
                int getValue();
            // read the value of the sequencer (how many
            // variation a component has)
                int readValue(){
                      return theValue;
            }
};

#endif  // __COMPONENT_H
```

component.cxx **********

```
#ifndef __DDBDEFINES_H
#include "ddbdefines.h"
#endif

#include <Type.h>
#include <Object.h>
#include <GlobalEntities.h>
#include <Database.h>
#include <Directory.h>

extern "C"
{
            char *getenv(const char *);
#include <strings.h>
}

#ifndef __COMPONENT_H
#include "component.h"
```

```
#endif

//extern userPtr;
extern char *thepath;
extern char *dirNamePtr;
extern COMPONENT* compPtr;

// ONTOS required constructor //
Sequencer::Sequencer(APL *theAPL):Object(theAPL)
{
}

//    Creates a Sequencer
Sequencer::Sequencer(char* name):Object(name)
{
          initDirectType((Type*)OC_lookup("Sequencer"));
          //directType(getDirectType());
          theValue=1;
}

Type* Sequencer::getDirectType()
{
          return (Type*)OC_lookup("Sequencer");
}

int Sequencer::getValue()
{
          theValue = theValue +1;
          return theValue;
}

COMPONENT::COMPONENT(APL *theAPL):
Object(theAPL)
{
}

//    Creates a list to hold text objects, then reset a
reference
//    to point to the list.

COMPONENT::COMPONENT(char* name,int var, int ver):
Object(name)
{
          initDirectType((Type*)OC_lookup("COMPONENT"));
          version_id = ver;
          variation_id = var;
```

```
          theName = Name();
          priv_author = 0;
          char *userPtr = getenv("USER");
          if (userPtr) {
                    priv_author = new char
[strlen(userPtr)+1];
                    if (priv_author)
                              strcpy(priv_author, userPtr);
          }

          DateCreated = setCreationDate();
          previous_version=0;
          next_version=0;
          part_of_list.Init(new

List((Type*)OC_lookup("COMPONENT")),this);
          sub_component_list.Init(new

List((Type*)OC_lookup("COMPONENT")),this);
          used_by_list.Init(new

List((Type*)OC_lookup("COMPONENT")),this);
          text_object_list.Init(new

List((Type*)OC_lookup("TEXT_OBJECT")),this);
}

// Member Functions //
//    returns the ONTOS Type for the COMPONENT class.

Type *COMPONENT::getDirectType()
{
          return (Type*)OC_lookup("COMPONENT");
}

void COMPONENT::putObject(OC_Boolean deallocate)
{
          //saves structure of the component lists
              ((List*)part_of_list.Binding(this))
                                        ->putObject(FALSE);
          ((List*)sub_component_list.Binding(this))
                                        ->putObject(FALSE);
          ((List*)used_by_list.Binding(this))-
>putObject(FALSE);
          ((List*)text_object_list.Binding(this))
                                                  -
```

```
>putObject(FALSE);
          // saves the component itself
          Object::putObject(deallocate);
}

void COMPONENT::deleteObject(OC_Boolean deallocate)
{
          //deletes structure of the component lists
              ((List*)part_of_list.Binding(this))
                    ->deleteObject(deallocate);
          ((List*)sub_component_list.Binding(this))
                    ->deleteObject(deallocate);
          ((List*)used_by_list.Binding(this))
                    ->deleteObject(deallocate);
          ((List*)text_object_list.Binding(this))
                    ->deleteObject(deallocate);
          // deletes the component itself
          Object::deleteObject(deallocate);
}


void COMPONENT::Destroy(OC_Boolean aborted)
{
          Entity* ent;
          ent = part_of_list.Binding(this);
          delete ent;
          ent = sub_component_list.Binding(this);
          delete ent;
          ent = used_by_list.Binding(this);
          delete ent;
          ent = text_object_list.Binding(this);
          delete ent;
          if (aborted) Object:
:
          Destroy(aborted);
}

// set creation time
time_t COMPONENT::setCreationDate()
{
          time_t *mytloc =0;
          time_t theTime;
          return theTime = time(mytloc);
}

//    delete the text objects of a COMPONENT object.
```

```
void COMPONENT::deleteComponentText()
{
          List *my_list =  TextObjectList();
          ListIterator my_iterator(my_list);
          TEXT_OBJECT *the_text_object;

          while(my_iterator.moreData())
          {
                    the_text_object =
                    (TEXT_OBJECT*)(Entity*)my_iterator();
                    the_text_object -> deleteObject();
          }
}
// check if the conponent name matches certain string
void COMPONENT::find_component(char *thename)
{
          char* nameptr=0;
          nameptr = CompnentName()+strlen(CompnentName())-
                    strlen(thename);
          if (strcmp( nameptr,thename)==0)
                    thepath=CompnentName();
}

void COMPONENT::find_a_component(char *thename)
{
          find_component(thename);
          if(thepath== 0)
          {
                    List *my_list =
                    (List*)sub_component_list.Binding(this);
                    ListIterator my_iterator(my_list);
                    COMPONENT *theComponent;
                    while(my_iterator.moreData())
                    {
                        theComponent =
                        (COMPONENT*)(Entity*)my_iterator();
                        theComponent-
>find_a_component(thename);
                    }
          }
}

void COMPONENT::find_parent(char *thename)
{
          List *my_list =
(List*)sub_component_list.Binding(this);
```

```
            ListIterator my_iterator(my_list);
            COMPONENT *theComponent;
            while(my_iterator.moreData())
            {
                        theComponent =
                        (COMPONENT*)(Entity*)my_iterator();
                        if(strcmp(theComponent
                        ->CompnentName(),thename)==0)
                        {
                                    compPtr= this;
                                    return;
                        }
                        else
                                    theComponent-
>find_parent(thename);
            }
}

//Displays the name of each text object in a COMPONENT object.
void COMPONENT::getComponentNames()
{
            cout << Name() <<"\n";
            List *my_list =
(List*)text_object_list.Binding(this);
            ListIterator my_iterator(my_list);
            TEXT_OBJECT *the_text_object;

            while(my_iterator.moreData())
            {
                        the_text_object =
(TEXT_OBJECT*)(Entity*)my_iterator();
                        the_text_object -> displayFileName();
            }
}

void COMPONENT::getComponentSubtreeNames()
{
            getComponentNames();
            List *my_list =
(List*)sub_component_list.Binding(this);
            ListIterator my_iterator(my_list);
            COMPONENT *theComponent;
            while(my_iterator.moreData())
            {
                        theComponent =
                        (COMPONENT*)(Entity*)my_iterator();
```

```
                    theComponent-
>getComponentSubtreeNames();
          }
}


Boolean COMPONENT::getComponentSource(char *fileMode)
{
          List *my_list =
(List*)text_object_list.Binding(this);
          ListIterator my_iterator(my_list);
          TEXT_OBJECT *the_text_object;
          Boolean write_failed = FALSE;
          while(my_iterator.moreData())
          {
                    the_text_object =
                    (TEXT_OBJECT*)(Entity*)my_iterator();
                    if (!the_text_object ->
                              rebuildTextFile(fileMode))
                              write_failed = TRUE;
          }
          if (write_failed)
                    return FAILED;
          else
                    return SUCCESS;
}

void COMPONENT::getComponentSubtreeSource(char *fileMode)
{
          getComponentSource(fileMode);
          List *my_list =
(List*)sub_component_list.Binding(this);
          ListIterator my_iterator(my_list);
          COMPONENT *theComponent;
          while(my_iterator.moreData())
          {
                    theComponent =
                    (COMPONENT*)(Entity*)my_iterator();
                    theComponent
                    ->getComponentSubtreeSource(fileMode);
          }
}


void COMPONENT::addTextObject(TEXT_OBJECT *my_text_object)
{
```

```
            List *my_list = TextObjectList();
            my_list -> Insert(my_text_object);
            my_list -> putObject();
}

void COMPONENT::replaceTextObject(TEXT_OBJECT
*my_text_object)
{
            OC_Boolean FOUND=FALSE;
            char *a_name=new char[strlen(my_text_object
                                    ->getFileName())+1];
            strcpy(a_name, my_text_object->getFileName());
            //  cout <<"new: " << a_name <<"\n";
            List *my_list = TextObjectList();
            ListIterator my_iterator(my_list);
            TEXT_OBJECT* a_text_object;
            while(my_iterator.moreData()&& !FOUND)
            {
                        a_text_object =
                        (TEXT_OBJECT*)(Entity*)my_iterator();
                        char *a_name1=new
char[strlen(a_text_object
                                    ->getFileName())+1];
                        strcpy(a_name1, a_text_object
                                    ->getFileName());
                        //      cout <<"old: " << a_name1 <<"\n";
                        if(strcmp(a_name, a_name1)==0){
                                    my_list->Remove(my_list
                                    -
>Index(a_text_object));
                                    my_list-
>Insert(my_text_object);
                                    FOUND = TRUE;
                        }
            }
            my_list -> putObject();
            putObject();
}

Boolean COMPONENT::getPSfile(char *fileMode)
{
            List *my_list =
(List*)text_object_list.Binding(this);
            ListIterator my_iterator(my_list);

            while(my_iterator.moreData())
```

```
        {
                TEXT_OBJECT *the_text_object =
                (TEXT_OBJECT*)(Entity*)my_iterator();
                char *the_file_name = the_text_object
                                ->getFileName();
                the_file_name=(the_file_name +
                    (strlen(the_text_object-
>getFileName())

-3));
                if(strcmp(the_file_name,".ps")==0)
                {
                        if (the_text_object
                        ->rebuildTextFile(fileMode));
                        return SUCCESS;
                }
        }
}


Boolean COMPONENT::getSPECfile(char *fileMode)
{

        List *my_list =
(List*)text_object_list.Binding(this);
        ListIterator my_iterator(my_list);
        while(my_iterator.moreData())
        {
                TEXT_OBJECT *the_text_object =
                (TEXT_OBJECT*)(Entity*)my_iterator();
                char *the_file_name = the_text_object
                                ->getFileName();
                the_file_name=(the_file_name +
                (strlen(the_text_object->getFileName())-
10));

if(strcmp(the_file_name,".spec.psdl")==0)
                {
                        if (the_text_object
                        ->rebuildTextFile(fileMode))
                                return SUCCESS;
                        else
                                return FAILED;
                }
        }
}
```

```
Boolean COMPONENT::getGRAPHfile(char *fileMode)
{
          List *my_list =
(List*)text_object_list.Binding(this);
          ListIterator my_iterator(my_list);
          while(my_iterator.moreData())
          {
                    TEXT_OBJECT *the_text_object =
                    (TEXT_OBJECT*)(Entity*)my_iterator();
                    char *the_file_name = the_text_object
                                        ->getFileName();
                    the_file_name=(the_file_name +
                    (strlen(the_text_object->getFileName())-
6));
                    if(strcmp(the_file_name,".graph")==0)
                    {
                              if (the_text_object
                                  -
>rebuildTextFile(fileMode))
                                        return SUCCESS;
                              else
                                        return FAILED;
                    }
          }
}


Boolean COMPONENT::getIMPfile(char *fileMode)
{
          List *my_list =
(List*)text_object_list.Binding(this);
          ListIterator my_iterator(my_list);
          while(my_iterator.moreData())
          {
                    TEXT_OBJECT *the_text_object =
                    (TEXT_OBJECT*)(Entity*)my_iterator();
                    char *the_file_name = the_text_object
                                        ->getFileName();
                    the_file_name=(the_file_name +
                    (strlen(the_text_object->getFileName())-
9));
                    if(strcmp(the_file_name,".imp.psdl")==0)
                    {
```

```
                                if (the_text_object
                                ->rebuildTextFile(fileMode))
                                        return SUCCESS;
                                else
                                        return FAILED;
                        }
                }
}


Boolean COMPONENT::getSOURCEfile(char *fileMode)
{
                List *my_list =
(List*)text_object_list.Binding(this);
                ListIterator my_iterator(my_list);
                while(my_iterator.moreData())
                {
                        TEXT_OBJECT *the_text_object =
                        (TEXT_OBJECT*)(Entity*)my_iterator();
                        char *the_file_name = the_text_object
                                        ->getFileName();
                        the_file_name=(the_file_name +
                        (strlen(the_text_object->getFileName())-
2));
                        if(strcmp(the_file_name,".a")==0)
                        {
                                if (the_text_object
                                ->rebuildTextFile(fileMode))
                                        return SUCCESS;
                                else
                                        return FAILED;
                        }
                }
}

char *COMPONENT::getTEXTPtr(char *TextType)
{

                List *my_list =
(List*)text_object_list.Binding(this);
                ListIterator my_iterator(my_list);
                while(my_iterator.moreData())
                {
                        TEXT_OBJECT *the_text_object =
                        (TEXT_OBJECT*)(Entity*)my_iterator();
                        char *the_file_name = the_text_object
```

```
                                        ->getFileName();
                    char *the_file=(the_file_name +
                        (strlen(the_text_object-
>getFileName())
                                        -
(strlen(TextType))));
                    if(strcmp(the_file,TextType)==0)
                    {
                            return the_text_object-
>text();
                    }
          }
          return (char *)0;
}

// add a subcomponent
void COMPONENT::addSubcomponent(COMPONENT* otherComponent)
{
          List *child_nodes = (List *)
                    sub_component_list.Binding(this);

          if (!this)
          {
                    cout << "<ERROR: cannot add a
subcomponent to a null component\n";
                    return;
          }
          if (!child_nodes)
          {
                    cout << "<ERROR: cannot add a null
subcomponent to a component\n";
                    return;
          }

          child_nodes->Insert(otherComponent);
          child_nodes->putObject();
          putObject();
}

void COMPONENT::replace_subconponent(COMPONENT* my_comp,
                                        char* comp_name)
{
          OC_Boolean FOUND=FALSE;
          char* salah=new char[strlen(comp_name)+1];
          strcpy(salah,comp_name);
          //        cout <<"comp: " << salah <<"\n";
```

```
          List *my_list = (List *)

sub_component_list.Binding(this);
          ListIterator my_iterator(my_list);
          while(my_iterator.moreData() && !FOUND){
                    COMPONENT* a_comp =
                    (COMPONENT*)(Entity*)my_iterator();
                    char* test=a_comp
                    ->CompnentName()+strlen(a_comp
                    ->CompnentName())-strlen(salah);
                    //          cout <<"old: " <<test <<"\n";
                    if(strcmp(test, salah)==0){
                              my_list->Remove(my_list
                                        ->Index(a_comp));
                              my_list->Insert(my_comp);
                              FOUND = TRUE;
                    }
          }
}

void COMPONENT::addParentComponent(COMPONENT*
otherComponent)
{
          List *parent_nodes = (List
*)part_of_list.Binding(this);
          if (!this)
          {
                    cout << "<ERROR: cannot add a
subcomponent to a null component\n";
                    return;
          }
          if (!parent_nodes)
          {
                    cout << "<ERROR: cannot add a null
subcomponent to a component\n";
                    return;
          }
          parent_nodes->Insert(otherComponent);
          parent_nodes->putObject();
          putObject();
}
```

comp_Operations.h **********

```
#ifndef __COMP_OPERATIONS_H
#define __COMP_OPERATIONS_H
```

```
#include"component.h"

char* get_last_token(char* comp_name);
char* get_red_of_extras(char* comp_name, char* a_name);
OC_Boolean set_New_prototype_Directory(char* aName);
OC_Boolean set_new_component_dir(char* aName);
OC_Boolean set_new_version_dir(char* aName, int var, int
ver);
void Add_SubComponent(char* dbName,char* name1,char*
a_name2,char* a_name3);
void CreateComponent(char* aName,int var, int ver);
void ShowComponent_subtree(char* dbName,char* name1, char*
aName);
void ShowComponent_subtree(char* name1, char* aName);
void ShowComponent(char* dbName,char* name1, char* aName);
void DeleteComponent(char* dbName,char* name1,char* aName);
void DumpComponent(char* name1,char* aName);
void CreatePrototype(char* dbName,char* aName);
void find_component_path(char* comp_name, char* a_name);
void Find_Parent(char* protoname,char*comp_name);
COMPONENT* Add_new_version (char* protoname,char*comp_name);
void DumpComponent_subtree(char* dbname,char*
protoname,char* comp_name);
void Show_prototypes(char* dbName);
void find_version_path(char* protoname,char*comp_name);
void Dump_version(char* dbname, char* protoname,char*
comp_name);
char* generate_new_configuration(char* protoname, COMPONENT*
my_comp);
void Dump_Imp_File(char* dbname, char* protoname,char*
comp_name);
void Dump_Spec_File(char* dbname, char* protoname,char*
comp_name);
void Dump_Imp_File1(char* protoname,char* comp_name);
void Dump_Spec_File1(char* protoname,char* comp_name);
void Show_component_versions(char* dbname,char*
protoname,char*comp_name);
#endif  //COMP_OPERATIONS_H
```

comp_Operations.cxx********

```
#include <Database.h>
#include <Directory.h>
#include <string.h>
#include <stream.h>
```

```
#include "text_object.h"
#include "component.h"
#include "My_String.h"
#include "comp_Operations.h"
#include "step_Operations.h"
// Globals
extern char *dirNamePtr;
extern  char* DESIGN_DATABASE_DIRECTORY;
extern  char* thepath ;
extern  char* v_path ;
extern COMPONENT* compPtr;

static char *my_ext[5]={".spec.psdl", ".imp.psdl", ".ps",
".graph",".a"};

char* get_last_token(char* comp_name)
{
            char *separator=">";
            char* word1;
            char* word2;
            word1=strtok(comp_name, separator);
            while(word1 !=NULL){
                 word2=word1;
                 word1=strtok(NULL, separator);
                }
return word2;
}

char* get_red_of_extras(char* comp_name, char* a_name)
{
            char *salah=new char[strlen(comp_name)+1];
            strcpy(salah,comp_name);
            char *badr= new char[strlen(a_name)+1];
            strcpy(badr,a_name);
            int n =strlen(salah)-2*strlen(badr)-4;
            char* word2=new char[n+1];
            strncpy(word2,salah,n);
          word2[n+1]='\0';
 return word2;

}

void CreatePrototype(char* dbname, char* aName)
{
        OC_open(dbname);
        OC_transactionStart();
```

```
        if (set_New_prototype_Directory(aName))
            CreateComponent( aName,1,1);
        OC_transactionCommit(OC_doNothing);
        OC_close();
}

void CreateComponent(char* aName,int var,int ver)
{
          cout << "Creating  a COMPONENT: " << aName << "
...\n";
        COMPONENT  *my_component = new
COMPONENT(aName,var,ver);
            int i;
          for(i=0; i<=4;++i){
             My_String temp1(My_String(dirNamePtr)+
          My_String("/")+My_String(aName)+
My_String(my_ext[i]));
           char *temp2= (char*) temp1;
           My_String temp3(My_String(aName)+
My_String(my_ext[i]));
          char *temp4= (char*) temp3;
          ifstream myFile(temp2); //=(char*)0;
            if(myFile){
                TEXT_OBJECT *a_text_obj;
                        a_text_obj= new TEXT_OBJECT();
                        a_text_obj->append(temp4, myFile);
                        my_component-
>addTextObject(a_text_obj);
              }
        }
          my_component->putObject();

}


void Add_SubComponent(char* dbname, char*
                      protoname,char*comp_name,char* parent)
{
      OC_open(dbname);
      OC_transactionStart();
      char temp[64];
      int ver, var;
      find_component_path(protoname,parent);
      if(thepath !=0){
      sscanf(comp_name,"%s %d:%d",temp,&var, &ver);
      My_String ptr1= My_String("^")+(My_String(thepath)-
```

```
                                        My_String(parent));
      char *ptr= (char*) ptr1;
          Directory* a_directory=(Directory*)OC_lookup(ptr);
      if(a_directory){
        OC_setWorkingDirectory(a_directory); //
(char*)the_dir);
        if(set_new_component_dir(comp_name)){
//          cout << "these are the values  " << ver << var
<<"\n";
          CreateComponent(temp,var,ver);
          COMPONENT *my_component1 = (COMPONENT*)
OC_lookup(temp);
          char* my_comp ="^";
          strcat(my_comp,thepath);
          COMPONENT *my_component2 = (COMPONENT*)
                                        OC_lookup(thepath);
             if(my_component1==NULL)
                cout <<" There is no such a component  \n";
            else{
                if(my_component2==NULL)
              cout <<" There is no such parent component  \n";
                else{
               my_component2->addSubcomponent(my_component1);
                my_component1-
>addParentComponent(my_component2);
                     }
                }
          }
          }
        else
           cout << "there is no such parent directory  \n";
      }
     else
        cout << "there is no such parent component   \n";
        OC_transactionCommit(OC_doNothing);
          OC_close();
}

COMPONENT* Add_new_version( char* protoname,char* p_name)
{
      char* comp_name= new char[strlen(p_name)+1];
      strcpy(comp_name,p_name);
      My_String temp3(My_String(comp_name)+
My_String("_seq"));
      char *temp2= (char*) temp3;
```

```
        int ver,var;
        find_component_path(protoname,comp_name);
        if(thepath !=0){
        My_String ptr1= My_String("^")+(My_String(thepath)-

My_String(comp_name));
        char *ptr= (char*) ptr1;
        COMPONENT *my_component1 = (COMPONENT*)
OC_lookup(thepath);
        if(my_component1 != NULL){
        ver = my_component1->versionNumber()+1;
        OC_setWorkingDirectory(ptr);
        if(my_component1->getNext_version()== 0)
           var = my_component1->variationNumber();
        else{
         Sequencer* my_sequencer=(Sequencer*)OC_lookup(temp2);
           if(my_sequencer!= NULL){
              var = my_sequencer->getValue();
              my_sequencer->putObject();
              }
           }
//       OC_setWorkingDirectory(ptr);
        if( set_new_version_dir(comp_name,var,ver)){
          CreateComponent(comp_name,var,ver);
          COMPONENT *my_component2 = (COMPONENT*)
                                  OC_lookup(comp_name);
          if(my_component1->getNext_version()== 0)
             my_component1->setNext_version(my_component2
                                          ->CompnentName());
             my_component1->putObject();
             my_component2->setPrevious_version(my_component1
                                          ->CompnentName());
             List* my_list = my_component1->subComponents();
             Iterator* my_iterator = my_list->getIterator();
             // For each item in the iterator
             while(my_iterator->moreData()) {
                COMPONENT* a_comp=(COMPONENT*)(Entity*)
                                          ((*my_iterator)());
                my_component2->addSubcomponent(a_comp);
               }
             my_component2->putObject();
             return my_component2;
             }
            }
           else return NULL;
        }
```

```
      else return NULL;

}

char*  generate_new_configuration(char* protoname,COMPONENT*
my_comp)
{
          int var, ver,var1,ver1;
          char temp[64];
          char temp2[64];
        char temp3[64];
        compPtr =NULL;
        char* badr= new char[strlen(my_comp-
>CompnentName())+1];
        strcpy(badr,my_comp->CompnentName());
        char* salah1=get_last_token(badr);
        char* comp_name1=new char[strlen(salah1)+1];
        strcpy(comp_name1,salah1);
        sscanf(protoname,"%s",temp);
        if (strcmp(comp_name1,temp)==0){
           var1 = my_comp->variationNumber();
           ver1 = my_comp->versionNumber();
           sprintf(temp3,"%s %d:%d",temp, var1, ver1);
            return temp3;
            }
        else{
          char* a_name= my_comp->getPrevious_version();
        Find_Parent(protoname,a_name);
        if(compPtr !=NULL){
           ver = compPtr->versionNumber()+1;
           char* ptr1=new char[strlen(compPtr-
>CompnentName())+1];
           strcpy(ptr1,compPtr->CompnentName());
           char* salah=get_last_token(ptr1);
           char* comp_name=new char[strlen(salah)+1];
           strcpy(comp_name,salah);
          My_String temp= My_String("^")+(My_String(compPtr
                    ->CompnentName())-My_String(comp_name));
           char* ptr=(char*) temp;
           Directory* my_dir=(Directory*)OC_lookup(ptr);
           if(my_dir){
           OC_setWorkingDirectory(my_dir);
           if(compPtr->getNext_version()== 0)
              var = compPtr->variationNumber();
           else{
              sprintf(temp2,"%s%s",comp_name,"_seq");
```

```
                Sequencer*
my_sequencer=(Sequencer*)OC_lookup(temp2);
                if(my_sequencer!= NULL){
                   var = my_sequencer->getValue();
                   my_sequencer->putObject();
                   }
                }
            if(set_new_version_dir(comp_name,var,ver))
            {
              COMPONENT  *my_component = new
                     COMPONENT(comp_name,var,ver);
                my_component->TextObjectList(compPtr
                                        ->TextObjectList());
              if(compPtr->getNext_version()== 0)
                 {
                 compPtr->setNext_version(my_component
                                        ->CompnentName());
                 compPtr->putObject();
                 }
              my_component->setPrevious_version(compPtr
                                        ->CompnentName());
            List* my_list = compPtr->subComponents();
            Iterator* my_iterator = my_list->getIterator();
            // For each item in the iterator
            while(my_iterator->moreData()) {
               COMPONENT* a_comp=(COMPONENT*)(Entity*)
                                        ((*my_iterator)());
               my_component->addSubcomponent(a_comp);
              }
              my_component-
>replace_subconponent(my_comp,comp_name1);
             my_component->putObject();
          generate_new_configuration(protoname,my_component);
            }
         }
       else  cout <<" no such dir:" << ptr <<"\n";
      }
     else cout << "no such comp: " << a_name <<"\n";
    }
}


void ShowComponent(char* dbname, char*
                     protoname,char*comp_name)
{
        OC_open(dbname);
```

```
        OC_transactionStart();
        find_component_path(protoname,comp_name);
        if(thepath !=0){
        COMPONENT *my_component= (COMPONENT*)
OC_lookup(thepath);
          if(my_component== NULL){
          cout << "Object: " << comp_name <<"    is not in
DDB...\n";
        }
        else{
             my_component->getComponentNames();
              }
        }
        OC_transactionCommit(OC_doNothing);
          OC_close();
}

void ShowComponent_subtree(char* dbname, char*
protoname,char*comp_name)
{
        OC_open(dbname);
        OC_transactionStart();
        find_component_path(protoname,comp_name);
        if(thepath !=0){
        COMPONENT *my_component= (COMPONENT*)
OC_lookup(thepath);
        if(my_component== NULL){
        cout << "Object: " << comp_name <<"    is not in
DDB...\n";
        }
        else{
           my_component->getComponentSubtreeNames();
            }
        }
//ShowComponent_subtree(protoname,comp_name);
        OC_transactionCommit(OC_doNothing);
          OC_close();
}


void DeleteComponent(char* dbname, char* protoname,
                                          char* comp_name)
{
        OC_open(dbname);
        OC_transactionStart();
        find_component_path(protoname,comp_name);
```

```
        if(thepath !=0){
        COMPONENT *my_component= (COMPONENT*)
OC_lookup(thepath);
        if(my_component== NULL)
        cout << "Object: " << comp_name <<"    is not in
DDB...\n";
        else{
           my_component->deleteComponentText();
           my_component->deleteObject(FALSE);
          }
          }
        else
           cout << "there is no such component  \n";
        OC_transactionCommit(OC_doNothing);
          OC_close();
}

void DumpComponent(char* protoname,char* comp_name)
{
        find_component_path(protoname,comp_name);
        if(thepath !=0){
        COMPONENT *my_component= (COMPONENT*)
OC_lookup(thepath);
        if(my_component== NULL)
        cout << "Object: " << comp_name <<"    is not in
DDB...\n";
        else{
           my_component->getComponentSource("w");
          }
        }
        else
           cout << "there is no such component  \n";
}

void Dump_Imp_File1(char* protoname,char* comp_name)
{
        find_component_path(protoname,comp_name);
        if(thepath !=0){
        COMPONENT *my_component= (COMPONENT*)
OC_lookup(thepath);
        if(my_component== NULL)
        cout << "Object: " << comp_name <<"    is not in
DDB...\n";
        else{
           my_component->getIMPfile("w");
          }
```

```
        }
        else
           cout << "there is no such component  \n";
}

void Dump_Spec_File1(char* protoname,char* comp_name)
{
        find_component_path(protoname,comp_name);
        if(thepath !=0){
        COMPONENT *my_component= (COMPONENT*)
OC_lookup(thepath);
        if(my_component== NULL)
        cout << "Object: " << comp_name <<"    is not in
DDB...\n";
        else{
           my_component->getSPECfile("w");
          }
        }
        else
           cout << "there is no such component  \n";
}

void DumpComponent_subtree(char* dbname, char*
                           protoname,char* comp_name)
{
        OC_open(dbname);
        OC_transactionStart();
        find_component_path(protoname,comp_name);
        if(thepath !=0){
        COMPONENT *my_component= (COMPONENT*)
OC_lookup(thepath);
        if(my_component== NULL)
        cout << "Object: " << comp_name <<"    is not in
DDB...\n";
        else
           my_component->getComponentSubtreeSource("w");

        }
        else
           cout << "there is no such component  \n";
        OC_transactionCommit(OC_doNothing);
          OC_close();
}

void Dump_version(char* dbname, char* protoname,char*
                                        comp_name)
```

```
{
        OC_open(dbname);
        OC_transactionStart();
        find_version_path(protoname,comp_name);
          if(v_path !=0){
//        cout << v_path << "\n";
        COMPONENT *my_component= (COMPONENT*)
OC_lookup(v_path);
        if(my_component== NULL)
        cout << "Object: " << comp_name <<"    is not in
DDB...\n";
        else
           my_component->getComponentSubtreeSource("w");

        }
        else
           cout << "there is no such component  \n";
        OC_transactionCommit(OC_doNothing);
        OC_close();
}

void Dump_Imp_File(char* dbname, char* protoname,char*
                                        comp_name)
{
        OC_open(dbname);
        OC_transactionStart();
        find_version_path(protoname,comp_name);
        if(v_path !=0){
//        cout << v_path << "\n";
        COMPONENT *my_component= (COMPONENT*)
OC_lookup(v_path);
        if(my_component== NULL)
        cout << "Object: " << comp_name <<"    is not in
DDB...\n";
        else
           my_component->getIMPfile("w");

        }
        else
           cout << "there is no such component  \n";
        OC_transactionCommit(OC_doNothing);
        OC_close();
}

void Dump_Spec_File(char* dbname, char* protoname,char*
                                        comp_name)
```

```
{
        OC_open(dbname);
        OC_transactionStart();
        find_version_path(protoname,comp_name);
        if(v_path !=0){
//        cout << v_path << "\n";
        COMPONENT *my_component= (COMPONENT*)
OC_lookup(v_path);
        if(my_component== NULL)
        cout << "Object: " << comp_name <<"    is not in
DDB...\n";
        else
           my_component->getSPECfile("w");

        }
        else
           cout << "there is no such component  \n";
        OC_transactionCommit(OC_doNothing);
        OC_close();
}

OC_Boolean set_New_prototype_Directory(char* aName)
{
//  char dir_name[64];
  Directory *prototype_dir = (Directory*)0;
  Directory *ddbRootDir = (Directory*)0;
  Directory *comp_dir=(Directory*)0;
  ddbRootDir = (Directory *)
OC_lookup(DESIGN_DATABASE_DIRECTORY);
  if(ddbRootDir)
      OC_setWorkingDirectory(ddbRootDir);
  else
    {
     ddbRootDir =  new Directory(DESIGN_DATABASE_DIRECTORY);
      ddbRootDir -> putObject();
      OC_setWorkingDirectory(ddbRootDir);
    }
        My_String temp(My_String(aName)+My_String("_dir"));
        char* dir_name=(char*) temp;
        prototype_dir= (Directory *) OC_lookup(dir_name);
        if (prototype_dir)
         {
          //OC_setWorkingDirectory(prototype_dir);
        cout << "Prototype: " << aName << "  already exist\n";
          return FALSE;
          }
```

```
        else {
           prototype_dir = new Directory(dir_name);
           prototype_dir -> putObject();
           OC_setWorkingDirectory(prototype_dir);
             My_String
temp1(My_String(aName)+My_String("_seq"));
           char* seq_name = (char*) temp1;
//           cout <<"Seq_name: " << seq_name <<"\n";
           Sequencer* my_sequencer= new Sequencer(seq_name);
           my_sequencer->putObject();
          My_String temp2(My_String(aName)+My_String("11"));
           char *p_dir = (char*)temp2;
           comp_dir = new Directory(p_dir);
           comp_dir -> putObject();
           OC_setWorkingDirectory(comp_dir);
           return TRUE;
           }
}

OC_Boolean set_new_component_dir(char* aName)
{
    int myver=0;
    int myvar=0;
     char temp[64];
     char temp1[64];
     char temp2[64];
     sscanf(aName,"%s %d:%d", temp,&myvar, &myver);
     sprintf(temp1,"%s%d%d", temp,myvar,myver);
     sprintf(temp2,"%s%s", temp, "_dir");
     Directory *comp_dir= (Directory *) OC_lookup(temp2);
     if(comp_dir)
        {
        //OC_setWorkingDirectory(comp_dir);
        cout << "component: " << temp <<"  already exist\n";
        return FALSE;
        }
     else
       {
        comp_dir = new Directory(temp2);
        comp_dir -> putObject();
        OC_setWorkingDirectory(comp_dir);
      // create a sequencer object
        char seq_name[64];
        sprintf(seq_name,"%s%s",temp,"_seq");
        Sequencer* my_sequencer= new Sequencer(seq_name);
        my_sequencer->putObject();
```

```
        //cout << "the new component_dir: " << temp2 <<"\n";
        Directory *version_dir= new Directory(temp1);
        version_dir -> putObject();
        OC_setWorkingDirectory(version_dir);
        return TRUE;
        }
}

OC_Boolean set_new_version_dir(char* aName, int var, int ver)
{
     char temp1[64];
     sprintf(temp1,"%s%d%d", aName,var,ver);
     Directory *version_dir= (Directory *) OC_lookup(temp1);
     if(version_dir)
        {
        OC_setWorkingDirectory(version_dir);
       cout << "version_dir: " << temp1 <<"  already exist\n";
        return FALSE;
        }
     else
        {
        version_dir = new Directory(temp1);
        version_dir -> putObject();
        OC_setWorkingDirectory(version_dir);
//        cout << "the new version_dir: " << temp1 <<"\n";
        return TRUE;
        }
}


void find_component_path(char* protoname,char*comp_name)
{
  char dir_name[64];
  char temp[64];
  thepath=0;
  int ver, var;
  Directory *prototype_dir = (Directory*)0;
  Directory *ddbRootDir = (Directory*)0;
  Directory *comp_dir=(Directory*)0;
  ddbRootDir = (Directory *)
OC_lookup(DESIGN_DATABASE_DIRECTORY);
  if(ddbRootDir)
    {
      OC_setWorkingDirectory(ddbRootDir);
    }
```

```
  else{
      cout << "there is no database as: "<<
DESIGN_DATABASE_DIRECTORY <<"\n";
      return;
       }
   sscanf(protoname, "%s %d:%d", temp, &var, &ver);
   sprintf(dir_name,"%s%s", temp,"_dir");
   prototype_dir= (Directory *) OC_lookup(dir_name);
   if (prototype_dir)
      {
       OC_setWorkingDirectory(prototype_dir);
       }
   else{
      cout << "there is no prototype  as: "<< dir_name <<"\n";
       return;
       }
   sprintf(dir_name,"%s%d%d", temp, var,ver);
   comp_dir= (Directory *) OC_lookup(dir_name);
   if(!comp_dir)
     {
      cout << "there is no such version as: "<< temp << var
<<ver <<"\n";
      return;
      }
    else{
    OC_setWorkingDirectory(comp_dir);
    COMPONENT* my_component=(COMPONENT*)OC_lookup(temp);
    my_component->find_a_component(comp_name);
    }
}

void Find_Parent(char* protoname,char*comp_name)
{
  char dir_name[64];
  char temp[64];
  thepath=0;
  int ver, var;
  Directory *prototype_dir = (Directory*)0;
  Directory *ddbRootDir = (Directory*)0;
  Directory *comp_dir=(Directory*)0;
  ddbRootDir = (Directory *)
OC_lookup(DESIGN_DATABASE_DIRECTORY);
  if(ddbRootDir)
    {
      OC_setWorkingDirectory(ddbRootDir);
    }
```

```
  else{
    cout<<"there is no database as:
"<<DESIGN_DATABASE_DIRECTORY <<"\n";
      return;
       }
   sscanf(protoname, "%s %d:%d", temp, &var, &ver);
   sprintf(dir_name,"%s%s", temp,"_dir");
   prototype_dir= (Directory *) OC_lookup(dir_name);
   if (prototype_dir)
      {
       OC_setWorkingDirectory(prototype_dir);
       }
   else{
      cout << "there is no prototype  as: "<< dir_name <<"\n";
       return;
       }
   sprintf(dir_name,"%s%d%d", temp, var,ver);
   comp_dir= (Directory *) OC_lookup(dir_name);
   if(!comp_dir)
     {
     cout << "there is no such version as: "<< temp << var
<<ver <<"\n";
       return;
      }
    else{
    OC_setWorkingDirectory(comp_dir);
    COMPONENT* my_component=(COMPONENT*)OC_lookup(temp);
    my_component->find_parent(comp_name);
    }
}

void find_version_path(char* protoname,char*comp_name)
{
          char temp[64];
          int ver, var;
          sscanf(comp_name, "%s %d:%d", temp, &var, &ver);
        find_component_path(protoname,temp);
        if(thepath !=0){
        My_String temp1(My_String(thepath)-My_String(temp));
        char* ptr=(char*) temp1;
        char the_comp[256];

sprintf(the_comp,"%s%s%s%s%d%d","^",ptr,">",temp,var,ver);
          My_String temp2(My_String(the_comp)+
                    My_String(">")+My_String(temp));
        v_path= (char*)temp2;
```

```
        }
          else
        cout <<"there is no such component: " << temp <<"\n";
}


void Show_prototypes(char* dbname)
{
        OC_open(dbname);
        OC_transactionStart();
          Directory *ddbRootDir = (Directory*)0;
          ddbRootDir = (Directory *)
                    OC_lookup(DESIGN_DATABASE_DIRECTORY);
        if(ddbRootDir==NULL)
        cout << "NO prototypes in DDB yet \n";
        else{
        OC_setWorkingDirectory(ddbRootDir);
          DirectoryIterator my_iterator(ddbRootDir);
          while(my_iterator.moreData()){
          Object* my_object = my_iterator();
          char* temp =0;
          temp= my_object->Name();
          char* temp1=temp + 9;
          char* temp2 = new char[strlen(temp1)-3];
          strncpy(temp2,temp1,(strlen(temp1)-4));
            temp2[strlen(temp1)-4]='\0';
            cout << temp2 << "\n";
          }
       }
          OC_transactionCommit(OC_doNothing);
        OC_close();
}

void Show_component_versions(char* dbname,char*
                              protoname,char*comp_name)
{
        OC_open(dbname);
        OC_transactionStart();
        find_component_path(protoname,comp_name);
        if(thepath !=0){
        My_String temp1(My_String("^")+(My_String(thepath)-
                              My_String(comp_name)));
        char* ptr=(char*) temp1;
        Directory *comp_dir=(Directory *) OC_lookup(ptr);
                if(comp_dir==NULL)
        cout << "NO such component in DDB yet \n";
```

```
        else{
        OC_setWorkingDirectory(comp_dir);
        DirectoryIterator my_iterator(comp_dir);
        while(my_iterator.moreData()){
          Object* my_object = my_iterator();
          char* temp =new char[strlen(my_object->Name())+1];
          strcpy(temp, my_object->Name());
          char* temp1=get_last_token(temp);
          My_String temp2=My_String(temp1)-2;
          char *temp3 =(char*) temp2;
          if(strcmp(temp3,comp_name)==0)
          cout << temp1 << "\n";
        }
      }
      }
        OC_transactionCommit(OC_doNothing);
        OC_close();
}
```

maincomp.cxx **********

```
#include <Database.h>
#include <Directory.h>
#include <stream.h>

extern "C"
{
#include <stdlib.h>
#include <stddef.h>
#include <string.h>
#include <ctype.h>
}

#include "comp_Operations.h"

// Globals
char *dirNamePtr = ".";
//char *dbName= "supportDB";
char* DESIGN_DATABASE_DIRECTORY="DesignDB";
char* thepath =(char*)0;
char* v_path =(char*)0;
COMPONENT* compPtr=NULL;

int main(int argc,char *argv[])
{
```

```
char *option=(char*)0;
char *aName=(char*)0;
char *aName1=(char*)0;
char *dbName=(char*)0;
char thename[64];
char theothername[64];

if (argv[1])
{
          dbName = new char[strlen(argv[1])+1];
          strcpy(dbName,argv[1]);
}

if (argv[2])
{
          option = new char[strlen(argv[2])+1];
          strcpy(option, argv[2]);
}
if(option[0] =='1'){
          // Create Prototype
              aName = new char[strlen(argv[3])+1];
          strcpy(aName,argv[3]);
          CreatePrototype(dbName, aName);
          //exit;
}
else if(option[0] == '2'){
          //ShowComponent
              aName = new char[strlen(argv[3])+1];
          strcpy(aName,argv[3]);
          aName1 = new char[strlen(argv[4])+1];
          strcpy(aName1,argv[4]);
          sprintf(thename,"%s %s",aName,aName1);
          aName1 = new char[strlen(argv[5])+1];
          strcpy(aName1,argv[5]);
          ShowComponent(dbName,thename,aName1);
          //exit;
}
else if(option[0] == '3'){
          // show subtree
              aName = new char[strlen(argv[3])+1];
          strcpy(aName,argv[3]);
          aName1 = new char[strlen(argv[4])+1];
          strcpy(aName1,argv[4]);
          sprintf(thename,"%s %s",aName,aName1);
          aName1 = new char[strlen(argv[5])+1];
          strcpy(aName1,argv[5]);
```

```
ShowComponent_subtree(dbName,thename,aName1);
                    //exit;
          }
          else if(option[0] == '4'){
                    //Add subComponent
                        aName = new char[strlen(argv[3])+1];
                    strcpy(aName,argv[3]);
                    aName1 = new char[strlen(argv[4])+1];
                    strcpy(aName1,argv[4]);
                    sprintf(thename,"%s %s",aName,aName1);
                    aName = new char[strlen(argv[5])+1];
                    strcpy(aName,argv[5]);
                    aName1 = new char[strlen(argv[6])+1];
                    strcpy(aName1,argv[6]);
                    sprintf(theothername,"%s
%s",aName,aName1);
                    aName1 = new char[strlen(argv[7])+1];
                    strcpy(aName1,argv[7]);
                    Add_SubComponent(dbName,thename,

theothername,aName1);
                    //exit;
          }
          else if(option[0] == '5'){
                    // Add new Version
                        aName = new char[strlen(argv[3])+1];
                    strcpy(aName,argv[3]);
                    aName1 = new char[strlen(argv[4])+1];
                    strcpy(aName1,argv[4]);
                    sprintf(thename,"%s %s",aName,aName1);
                    aName1 = new char[strlen(argv[5])+1];
                    strcpy(aName1,argv[5]);
                    Add_new_version(thename,aName1);
                    //exit(0);
          }
          else if(option[0] == '6'){
                    // Dump component
                    aName = new char[strlen(argv[3])+1];
                    strcpy(aName,argv[3]);
                    aName1 = new char[strlen(argv[4])+1];
                    strcpy(aName1,argv[4]);
                    sprintf(thename,"%s %s",aName,aName1);
                    aName1 = new char[strlen(argv[5])+1];
                    strcpy(aName1,argv[5]);
                    DumpComponent(thename,aName1);
```

```
        }
        else if(option[0] == '7'){
                aName = new char[strlen(argv[3])+1];
                strcpy(aName,argv[3]);
                aName1 = new char[strlen(argv[4])+1];
                strcpy(aName1,argv[4]);
                sprintf(thename,"%s %s",aName,aName1);
                aName1 = new char[strlen(argv[5])+1];
                strcpy(aName1,argv[5]);

DumpComponent_subtree(dbName,thename,aName1);
        }
        else if(option[0] == '8'){
                Show_prototypes(dbName);
        }
        else if(option[0] == '9'){
                aName = new char[strlen(argv[3])+1];
                strcpy(aName,argv[3]);
                //aName1 = new char[strlen(argv[4])+1];
                //strcpy(aName1,argv[4]);
                sprintf(thename,"%s
%s:%s",aName,"1","1");
                aName = new char[strlen(argv[4])+1];
                strcpy(aName,argv[4]);
                aName1 = new char[strlen(argv[5])+1];
                strcpy(aName1,argv[5]);
                sprintf(theothername,"%s
%s",aName,aName1);

Dump_version(dbName,thename,theothername);
        }
        else if(option[0] == 'a'){
                aName = new char[strlen(argv[3])+1];
                strcpy(aName,argv[3]);
                //aName1 = new char[strlen(argv[4])+1];
                //strcpy(aName1,argv[4]);
                sprintf(thename,"%s
%s:%s",aName,"1","1");
                aName = new char[strlen(argv[4])+1];
                strcpy(aName,argv[4]);
                aName1 = new char[strlen(argv[5])+1];
                strcpy(aName1,argv[5]);
                sprintf(theothername,"%s
%s",aName,aName1);

Dump_Spec_File(dbName,thename,theothername);
```

```
        }
        else if(option[0] == 'b'){
                aName = new char[strlen(argv[3])+1];
                strcpy(aName,argv[3]);
                //aName1 = new char[strlen(argv[4])+1];
                //strcpy(aName1,argv[4]);
                sprintf(thename,"%s
%s:%s",aName,"1","1");
                aName = new char[strlen(argv[4])+1];
                strcpy(aName,argv[4]);
                aName1 = new char[strlen(argv[5])+1];
                strcpy(aName1,argv[5]);
                sprintf(theothername,"%s
%s",aName,aName1);

Dump_Imp_File(dbName,thename,theothername);
        }
        else if(option[0] == 'c'){
                // show subtree
                    aName = new char[strlen(argv[3])+1];
                strcpy(aName,argv[3]);
                aName1 = new char[strlen(argv[4])+1];
                strcpy(aName1,argv[4]);
                sprintf(thename,"%s %s",aName,aName1);
                aName1 = new char[strlen(argv[5])+1];
                strcpy(aName1,argv[5]);
                Show_component_versions(dbName,thename,
                                                aName1);
                //exit;
        }
        else
                cout <<"Wrong Option: " << option << "
Try again \n";
}
```

3. Class Designer

person.h **********

```
#include <Reference.h>
#include <Object.h>
#include <List.h>


class Person : public Object
```

```
{
  private:
    int       priv_level;
    int       priv_status;

  public:

    // Constructors
    Person(char* name=(char*)0,int level= 0, int status=0);
    Person (APL*);

    // Get direct type
    Type *getDirectType();

    // Accessors
    int     PersonLevel();
    void    PersonLevel(int level);
    int     PersonStatus();
    void    PersonStatus(int status);
    void    display();
};
```

person.cxx **********

```
#include "person.h"
#include <Directory.h>
#include <Object.h>
#include <stream.h>

//-------------------------------------------------
// constructors
//-------------------------------------------------

Person::Person(APL *theAPL) :
Object(theAPL)
{
}

Person::Person(char* name,int level, int status):
Object(name)

{
          initDirectType((Type *)OC_lookup("Person"));
          priv_level = level;
          priv_status= status;
```

```
}


//-------------------------------------------------
// accessors
//-------------------------------------------------

Type *Person::getDirectType()
{
          return (Type*)OC_lookup("Person");
}


void  Person::PersonLevel(int level)
{
          priv_level = level;
}

int Person::PersonLevel()
{
          return  priv_level;
}


void Person::PersonStatus(int status)
{
          priv_status= status;
}
int Person::PersonStatus()
{
          return priv_status;
}
```

SetOperations.h **********

```
#include <Database.h>
#include <Directory.h>          // for Object naming
#include <Set.h>                // for Set class
#include <stream.h>
#include <string.h>
#include <stdio.h>
#include "person.h"

          void trivial();
          void addDesigner(char* dbName,char* aName,int
Level);
```

```
        void showDesigners(char* dbName);
        void deleteDesigner(char* dbName,char* aName);
        void changeExpLevel(char* dbName,char* aName, int
                                                  Level);
        void changeStatus(char* dbName,char* aName);
        void showDesigner(char* dbName,char* aName);
```

SetOperations.cxx **********

```
#include <Database.h>
#include <Directory.h>          // for Object naming
#include <Set.h>                // for Set class
#include <List.h>                // for Set class
#include <stream.h>
#include <stdio.h>
#include <Type.h>
#include <Object.h>
#include "SetOperations.h"


//static char *levels[3] = {"Low", "Medium", "High"};
//static char *States[2] = {"Free","Busy"};

extern char *levels[3];
extern char *States[2];

// Add designer
void addDesigner(char* dbName,char* aName,int Level)
{
  char personString[64];
  OC_open(dbName);
  OC_transactionStart();
  // Create designer objects and insert into set
    Person *aPerson = (Person*)OC_lookup(aName);
    if ( aPerson == NULL ) {
//        cout << "Creating designer object: "<< aName << "
...\n";
        // Create a designer object
        aPerson = new Person(aName, Level, 0);
        // Put it in the database
        aPerson->putObject();
    }else
          cout << "Designer object: "<< aName << "  already
```

```
exist...\n";

  OC_transactionCommit();
  OC_close();
}

void showDesigners(char* dbName)
{
  OC_open(dbName);
  OC_transactionStart();
  InstanceIterator it ((Type*) OC_lookup("Person"));
  while (it.moreData()) {
          Person* nextPerson = (Person*) (Entity*)it();
          cout.width(24);
        cout.setf(ios::left,ios::adjustfield);
        cout << nextPerson->Name();
        cout.width(19);
        cout.setf(ios::left,ios::adjustfield);
        cout <<levels[nextPerson->PersonLevel()];
        cout <<States[nextPerson->PersonStatus()] <<"\n";
     }
    it.Destroy();
  OC_transactionCommit();
  OC_close();
}


// Delete designer
void deleteDesigner(char* dbName,char* aName)
{
  OC_open(dbName);
  OC_transactionStart();
      // Get the item
    Person *aPerson = (Person*)OC_lookup(aName);
    if ( aPerson == NULL )
      cout << " Designer: " << aName << "  not in the Database
\n";
        // Create a designer object
    else {
      // Print out its name
 //     cout << "Removing "<< aPerson->Name() << "\n";
      // Remove it
      aPerson->deleteObject();
     }
   OC_transactionCommit();
   OC_close();
```

```
}


// Change the designer's expertise level
void changeExpLevel(char* dbName,char* aName, int Level)
{
    OC_open(dbName);
    OC_transactionStart();
    Person *aPerson = (Person*)OC_lookup(aName);
    if ( aPerson == NULL )
      cout << " Designer: " << aName << "  not in the Database
\n";
        // change his expertise level
    else {
//    cout <<"Designer's old Expertise = "<<levels[aPerson-
>PersonLevel()];
//    cout << "\n";
      // set the new level
      aPerson->PersonLevel( Level);
      // Put it in the database
      aPerson->putObject();
    }
   OC_transactionCommit();
   OC_close();
}

void changeStatus(char* dbName,char* aName)
{
    OC_open(dbName);
    OC_transactionStart();
    int  S = 0;
    Person *aPerson = (Person*)OC_lookup(aName);
    if ( aPerson == NULL )
      cout << " Designer: " << aName << "  not in the Database
\n";
    else {
        // change his status
        if (aPerson->PersonStatus()==0)
            S = 1;
        else {
       if (aPerson->PersonStatus()== 1)
           S = 0;
       }
      // set the new status
      aPerson->PersonStatus(S);
      // Put it in the database
```

```
      aPerson->putObject();
    }
   OC_transactionCommit();
   OC_close();
}
```

maindes.cxx **********

```
#include <string.h>
#include "SetOperations.h"
// Globals
char *levels[3] = {"Low", "Medium", "High"};
char *States[2] = {"Free","Busy"};

int main(int argc,char *argv[])
{
char *option=(char*)0;
char *aName=(char*)0;
char *dbName=(char*)0;
char *mylevel=(char*)0;
int  Level=0;
  if (argv[1])
    {
      dbName = new char[strlen(argv[1])+1];
      strcpy(dbName,argv[1]);
    }

  if (argv[2])
    {
      option = new char[strlen(argv[2])+1];
      strcpy(option, argv[2]);
    }

     if(option[0] =='1'){
          //Add designer
        aName = new char[strlen(argv[3])+1];
        strcpy(aName,argv[3]);
        mylevel= new char[strlen(argv[4])+1];
          strcpy(mylevel,argv[4]);
        Level=(int) mylevel[0]- 48;
        addDesigner(dbName, aName, Level);
        //exit;
          }
     else if(option[0] == '2'){
        showDesigners(dbName);
        //exit;
```

```
        }
      else if(option[0] == '3'){
           aName = new char[strlen(argv[3])+1];
         strcpy(aName,argv[3]);
         deleteDesigner(dbName, aName);
         //exit;
         }
      else if(option[0] == '4'){
         aName = new char[strlen(argv[3])+1];
         strcpy(aName,argv[3]);
         mylevel= new char[strlen(argv[4])+1];
         strcpy(mylevel,argv[4]);
         Level=(int) mylevel[0]- 48;
         changeExpLevel(dbName, aName, Level);
         //exit;
         }
       else if(option[0] == '5'){
         aName = new char[strlen(argv[3])+1];
         strcpy(aName,argv[3]);
           changeStatus(dbName, aName);
         //exit(0);
         }
}
```

4. Class Assignment

sched.h **********

```
#include <Reference.h>
#include <Object.h>
#include <List.h>
#include "support_classes.h"

class Schedule :
public Object
{
private:
          char*     priv_designer;
          Time      priv_finish;
          Time      priv_start;

public:

          // Constructors
              Schedule(char* aName=(char*)0,char*
                                        designer=(char*)0);
```

```
        Schedule (APL*);

        // Get direct type
            Type *getDirectType();

        // Accessors
            char*   AssignedDesigner();
        void    AssignedDesigner(char* designer);
        Time AssignmentStart();
        void AssignmentStart(Time EstimatedStart);
        Time AssignmentFinish();
        void AssignmentFinish(Time EstimatedFinish);
};
```

sched.cxx **********

```
#include <Directory.h>
#include <Object.h>
#include <stream.h>
#include "sched.h"
#include "support_classes.h"

//-----------------------------------------------
// constructors for the assignment class
//-----------------------------------------------

Schedule::Schedule(APL *theAPL) :
Object(theAPL),priv_start((APL*)0), priv_finish((APL*)0)
{
}

Schedule::Schedule(char* aName,char* designer):
Object(aName),
priv_start(0,0,0,0,0), priv_finish(0,0,0,0,0)

{
        initDirectType((Type *)OC_lookup("Schedule"));
        priv_designer = designer;
}


//-----------------------------------------------
// accessors
//-----------------------------------------------

Type *Schedule::getDirectType()
```

```
{
        return (Type*)OC_lookup("Schedule");
}


void  Schedule::AssignedDesigner(char* designer)
{
        priv_designer = designer;
}

char*Schedule :: AssignedDesigner()
{
        return  priv_designer;
}

Time Schedule:: AssignmentStart()
{
        return priv_start;
}

void Schedule:: AssignmentStart(Time EstimatedStart)
{
        priv_start= EstimatedStart;
}

Time Schedule:: AssignmentFinish()
{
        return priv_finish;
}

void Schedule:: AssignmentFinish(Time EstimatedFinish)
{
        priv_finish= EstimatedFinish;
}
```

schedOp.h **********

```
#ifndef __SCHEDOP_H
#define __SCHEDOP_H
void addAssignment(char *dbname,char* listName, char*
step_id,
                char* date,int start,int finish,char* aName);
void showSchedule(char *dbname,char* listName);
void update_start_time(char *dbname,char* listName, char*
                                                designer);
void getSchedule(char *dbname,char* listName);
```

```
void deleteSchedule(char *dbname,char* listName);
void deleteAssignment(char *dbname,char* listName,char*
step_id);
void deleteAssignment1(char* listName,char* step_id);
void getSchedule_1(char *dbname,char* listName,char*
cur_time);
#endif
```

schedOp.cxx **********

```
#include <Database.h>
#include <Directory.h>          // for Object naming
#include <Set.h>                // for Set class
#include <List.h>                // for Set class
#include <stream.h>
#include <stdio.h>
#include <Type.h>
#include <Object.h>
#include <string.h>

#include "step.h"
#include "sched.h"
#include "support_classes.h"
#include "schedOp.h"


// Add Assignment
void addAssignment(char *dbname,char* listName,char* Mystep,
char* Mydate,int start,int finish, char* MyD_name)
{
          OC_open(dbname);
          OC_transactionStart();
          // Create assignment object and insert into list
              List *aList = (List *)OC_lookup(listName);
          // If it does not exist, create it
              if(aList  == NULL) {
                    cout << "Creating list object ...\n";
                    // Create a new list called MySchedule
                        aList  = new List
                    ((Type*)OC_lookup("Schedule"),
listName);
          }
          //cout << aList->Name() << " already exists.\n";
          Schedule *aSchedule =
(Schedule*)OC_lookup(Mystep);
          if ( aSchedule == NULL )
```

```
          {
                    //cout << "Creating record object: "<<
Mystep << " ...\n";
                    // Create a designer object
                    aSchedule = new
Schedule(Mystep,MyD_name);
                    Time startTime(Mydate);
                    startTime=startTime+start;
                    Time finishTime(Mydate);
                    finishTime=finishTime+finish;
                    aSchedule-> AssignmentStart(startTime);
                    aSchedule->
AssignmentFinish(finishTime);
                    // Put it in the database
                        aSchedule->putObject();
                    // Check to see if object is already in
list
                       // This is necessary because if object
                       // is already in list, the Insert  will
add
                        // it again to the list
                        if (aList->isMember(aSchedule) ==
FALSE) {
                              cout << "Inserting " <<
aSchedule->Name() << " into list...\n";
                              // Insert object into list
                                  aList->Insert(aSchedule);
                              aList->putObject();
                    }
          }
          else
          {
                    Time startTime(Mydate);
                    startTime=startTime+start;
                    Time finishTime(Mydate);
                    finishTime=finishTime+finish;
                    aSchedule-> AssignmentStart(startTime);
                    aSchedule->
AssignmentFinish(finishTime);
                    aSchedule->AssignedDesigner(MyD_name);
                    // Put it in the database
                        aSchedule->putObject();
          }

          OC_transactionCommit();
          OC_close();
```

```
}

void update_start_time(char *dbname,char* listName, char*
designer)
{
          OC_open(dbname);
          OC_transactionStart();
          OC_Boolean FOUND=FALSE;
          // Create assignment object and insert into list
              List *aList = (List *)OC_lookup(listName);
          // If it does not exist, create it
              if(aList  != NULL) {
                    Iterator* anIterator = aList-
>getIterator();

                    // For each item in the iterator
                        while(anIterator->moreData()&&
!FOUND) {
                              Schedule* nextAssignment =
                         (Schedule*)
(Entity*)((*anIterator)());
                              if(strcmp(nextAssignment
                              -
>AssignedDesigner(),designer)==0){
                                        FOUND = TRUE;
                                        nextAssignment->

AssignmentStart(nextAssignment->

AssignmentStart()+2);
                                        nextAssignment
                                        ->putObject();
                              }
                    }
          }
          OC_transactionCommit();
          OC_close();
}


void showSchedule(char *dbname,char* listName)
{
          OC_open(dbname);
          OC_transactionStart();
          List *aList = (List *)OC_lookup(listName);
          if(aList  == NULL) {
```

```
                    cout << "No Available Schedule ...\n";
                    OC_transactionCommit();
                    OC_close();
                    return;

          }
          if(aList->Cardinality()==0)
                    cout << "No Available Schedule ...\n";
          Iterator* anIterator = aList->getIterator();

          // For each item in the iterator
              while(anIterator->moreData()) {
                    Schedule* nextAssignment = (Schedule*)
                    (Entity*)((*anIterator)());
                    cout.width(10);
                    cout.setf(ios::left,ios::adjustfield);
                    cout << nextAssignment->Name();
                    cout.width(20);
                    cout.setf(ios::left,ios::adjustfield);
                    cout << nextAssignment->

AssignmentStart().makeString();
                    cout.width(20);
                    cout.setf(ios::left,ios::adjustfield);
                    cout <<nextAssignment->
                    AssignmentFinish().makeString();
                    cout << nextAssignment-
>AssignedDesigner()
                         << "\n";
          }
          delete anIterator;
          OC_transactionCommit();
          OC_close();
}

void getSchedule(char *dbname,char* listName)
{
          OC_open(dbname);
          OC_transactionStart();
          int T2,T3;
          List *aList = (List *)OC_lookup(listName);
          if(aList  == NULL) {
                    cout << "No Such Schedule ...\n";
                    OC_transactionCommit();
                    OC_close();
                    return;
```

```
        }
        Schedule* first_ass=(Schedule*)aList
                            ->getEntityElement(0);
        Time T1=first_ass->AssignmentStart();

        Iterator* anIterator = aList->getIterator();

        // For each item in the iterator
            while(anIterator->moreData()) {
                  Schedule* nextAssignment = (Schedule*)
                            (Entity*)((*anIterator)());
                  cout.width(10);
                  cout.setf(ios::left,ios::adjustfield);
                  cout << nextAssignment->Name();
                  cout.width(20);
                  cout.setf(ios::left,ios::adjustfield);
                  T2=nextAssignment-> AssignmentStart()-
T1;
                  cout << T2;
                  cout.width(20);
                  cout.setf(ios::left,ios::adjustfield);
                  T3=nextAssignment-> AssignmentFinish()-
T1;
                  cout <<T3;
                  cout << nextAssignment-
>AssignedDesigner()
                       << "\n";
        }
        delete anIterator;
        OC_transactionCommit();
        OC_close();
}

void getSchedule_1(char *dbname,char* listName,char*
cur_time)
{
        OC_open(dbname);
        OC_transactionStart();
        List *aList = (List *)OC_lookup(listName);
        if(aList  == NULL) {
                  //   cout << "No Such Schedule ...\n";
                  OC_transactionCommit();
                  OC_close();
                  return;
        }
```

```
        Time T1(cur_time);

        Iterator* anIterator = aList->getIterator();

        // For each item in the iterator
            while(anIterator->moreData()) {
                Schedule* nextAssignment = (Schedule*)
                        (Entity*)((*anIterator)());
                if(T1 > nextAssignment-
>AssignmentStart()){
                        cout.width(10);
                cout.setf(ios::left,ios::adjustfield);
                        cout << nextAssignment->Name()
                             <<"\n";
                        cout.width(20);
                cout.setf(ios::left,ios::adjustfield);
                        int T3=nextAssignment->
                                AssignmentFinish()-
T1;
                        cout <<T3 <<"\n";
                        cout << nextAssignment
                        ->AssignedDesigner() << "\n";
                }
        }
        delete anIterator;
        OC_transactionCommit();
        OC_close();
}

void deleteSchedule(char *dbname,char* listName)
{
        OC_open(dbname);
        OC_transactionStart();
        List *aList = (List *)OC_lookup(listName);
        if(aList  == NULL) {
                OC_transactionCommit();
                OC_close();
                return;
        }
        //Delete the Schedule List
            aList->deleteCluster();
        OC_transactionCommit();
        OC_close();
}

void deleteAssignment(char *dbname,char* listName, char*
```

```
step_id)
{
        OC_open(dbname);
        OC_transactionStart();
        OC_Boolean FOUND=FALSE;
        List *aList = (List *)OC_lookup(listName);
        if(aList  == NULL) {
                cout << "Nothing to delete ...\n";
                return;
        }

        Iterator* anIterator = aList->getIterator();

        // For each item in the iterator
            while(anIterator->moreData()&& !FOUND) {
                Schedule* nextAssignment = (Schedule*)
                        (Entity*)((*anIterator)());
                if (strcmp(nextAssignment->Name(),
                                step_id)==0){
                        aList->Remove(aList
                                -
>Index(nextAssignment));
                        nextAssignment-
>deleteObject();
                        FOUND = TRUE;
                }
        }
        if (aList->Cardinality() == 0)
                aList->deleteObject();
        else
                aList->putObject();
        OC_transactionCommit();
        OC_close();
}

void deleteAssignment1(char* listName, char* step_id)
{
        OC_Boolean FOUND=FALSE;
        List *aList = (List *)OC_lookup(listName);
        if(aList  == NULL) {
                //      cout << "Nothing to delete ...\n";
                return;
        }

        Iterator* anIterator = aList->getIterator();
```

```
        // For each item in the iterator
            while(anIterator->moreData()&& !FOUND) {
                Schedule* nextAssignment = (Schedule*)
                        (Entity*)((*anIterator)());
                if (strcmp(nextAssignment->Name(),
                                    step_id)==0){
                        aList->Remove(aList
                                    -
>Index(nextAssignment));
                        nextAssignment-
>deleteObject();
                        FOUND = TRUE;
                }
        }
        aList->putObject();
}
```

mainsched.cxx **********

```
#include <Database.h>
#include <Directory.h>          // for Object naming
#include <Set.h>                // for Set class
#include <stream.h>
#include <stdio.h>
#include <string.h>

#include "sched.h"
#include "support_classes.h"
#include "schedOp.h"
int main(int argc,char *argv[])
{
        char *option=(char*)0;
        char *aName=(char*)0;
        char *aName1=(char*)0;
        char *aName2=(char*)0;
        char *Dname=(char*)0;
        char *dbName=(char*)0;
        char date1[64];
        char *listName="MySchedule";
        int v1,v2;

        if (argv[1])
        {
                dbName = new char[strlen(argv[1])+1];
                strcpy(dbName,argv[1]);
```

```
        }

        if (argv[2])
        {
                option = new char[strlen(argv[2])+1];
                strcpy(option, argv[2]);
        }

        if(option[0] =='1'){
                //Add designer
                    aName = new char[strlen(argv[3])+1];
                strcpy(aName,argv[3]);
                Dname= new char[strlen(argv[4])+1];
                strcpy(Dname,argv[4]);
                aName1= new char[strlen(argv[5])+1];
                strcpy(aName1,argv[5]);
                aName2= new char[strlen(argv[6])+1];
                strcpy(aName2,argv[6]);
                sprintf(date1,"%s %s",aName1,aName2);
                aName1= new char[strlen(argv[7])+1];
                strcpy(aName1,argv[7]);
                sscanf(aName1,"%d",&v1);
                aName2= new char[strlen(argv[8])+1];
                strcpy(aName2,argv[8]);
                sscanf(aName2,"%d",&v2);
                addAssignment(dbName, listName,aName,
date1,
                                                    v1,v2,
Dname);
        }
        else if(option[0] == '2'){
                showSchedule(dbName, listName);
        }
        else if(option[0] == '3'){
                deleteSchedule(dbName, listName);
        }
        else if(option[0] == '4'){
                aName = new char[strlen(argv[3])+1];
                strcpy(aName,argv[3]);
                deleteAssignment(dbName,
listName,aName);
        }
        else if(option[0] == '5'){
                getSchedule(dbName, listName);
        }
        else if(option[0] == '6'){
```

```
                    aName = new char[strlen(argv[3])+1];
                    strcpy(aName,argv[3]);
                    Dname= new char[strlen(argv[4])+1];
                    strcpy(Dname,argv[4]);
                    sprintf(date1,"%s %s",aName,Dname);
                    getSchedule_1(dbName, listName,date1);
          }
          else if(option[0] == '7'){
                    aName = new char[strlen(argv[3])+1];
                    strcpy(aName,argv[3]);
                    update_start_time(dbName,
listName,aName);
          }
}
```

5. Class Time

support_classes.h **********

```
#ifndef __SUPPORT_CLASSES_H
#define __SUPPORT_CLASSES_H


class Time{

    private:
        unsigned  priv_minuteInHour :11;
        unsigned  priv_hourInDay    :5;
        unsigned  priv_dayInMonth   :5;
        unsigned  priv_monthInYear  :4;
        unsigned  priv_yearFrom1993 :5;
    public:
      Time( int min, int hour, int day, int month, int year );
        Time(char* dateTimeString);
        Time( APL* theAPL);

        Time operator+( int duration );
        int  operator-(Time& anotherTime);
        OC_Boolean operator==( Time& anotherTime );
        OC_Boolean operator>( Time& anotherTime );
        char* makeString( );
        void  display();
};

#endif  //__SUPPORT_CLASSES_H
```

support_classes.cxx **********

```
#include <Type.h>
#include <Object.h>
#include <GlobalEntities.h>
#include <Database.h>
#include <Directory.h>
#include <stream.h>
extern "C"
{
#include <strings.h>
#include <ctype.h>
#include <stddef.h>
#include <string.h>
}

#ifndef __SUPPORT_CLASSES_H
#include "support_classes.h"
#endif

// Init nullTine
static Time nullTime(0,0,0,0,0);
// Constructor used for + operation, just initialize fields
Time::Time( int min, int hour, int day, int month, int year ):
priv_minuteInHour( min ), priv_hourInDay ( hour),
priv_dayInMonth ( day ), priv_monthInYear( month ),
priv_yearFrom1993( year )
{
}

// Activation Constructor
Time ::Time(APL*)
{
}

// Constructor used by end user, converts standard
// mm/dd/yy hh:
mm format to internal representation
Time::Time( char* dateTimeString )
{
          int month, day, year, hour, minute;

          sscanf( dateTimeString, "%d/%d/%d %d:%d", &month,
&day,
                                        &year,
```

```
&hour,&minute);
          priv_yearFrom1993 = year-93;
          priv_monthInYear  = month;
          priv_dayInMonth   = day;
          priv_hourInDay    = hour;
          priv_minuteInHour  = minute;
}

// Adds duration to the hourInDay field,
Time Time:
:
operator+( int duration )
{
          //        unsigned t1, t2, t3, t4,t5;
          int t1, t2, t3, t4,t5;
          t4 = priv_hourInDay;
          t4 = t4+duration;
          t1=t4;
          priv_hourInDay=t4;
          if (t4 > 16) {
                    t4 = (t4-16)%8 + 8;
                    priv_hourInDay=t4;

                    t5=priv_dayInMonth;
                    t5 = (t5 +1+ (t1-16)/8);
                    priv_dayInMonth=t5;
                    if (t5 > 30){
                              t5= t5- 30;
                              priv_dayInMonth=t5;

                              t3=priv_monthInYear;
                              t3= t3+1 ;
                              priv_monthInYear=t3;
                              if (t3>12){
                                        t3=t3-12;
                                        priv_monthInYear=t3;

t2=priv_yearFrom1993;
                                        t2=t2+1;

priv_yearFrom1993=t2 ;
                              }
                    }
          }
          Time result( priv_minuteInHour,priv_hourInDay,
```

```
priv_dayInMonth,priv_monthInYear,priv_yearFrom1993 );
          return result;
}

// subtract two times resturning the difference in hours,
int Time::operator-(Time& my_time  )
{
          int result;
          int month, day, year, hour;
          hour=priv_hourInDay-my_time.priv_hourInDay;
          day= priv_dayInMonth-my_time.priv_dayInMonth;
          month= priv_monthInYear-my_time.priv_monthInYear;
          year=priv_yearFrom1993-my_time.priv_yearFrom1993;

          result=hour+(day*8)+(month*240)+(year*12*240);
          return result;
}


OC_Boolean Time::operator==( Time& anotherTime )
{
          return (OC_Boolean) (*(int*)this ==
*(int*)&anotherTime);
}

OC_Boolean Time::operator>( Time& anotherTime )
{
    if (priv_yearFrom1993 > anotherTime.priv_yearFrom1993)
        return TRUE;
    else{
       if(priv_yearFrom1993 == anotherTime.priv_yearFrom1993
&&
            priv_monthInYear > anotherTime.priv_monthInYear)
            return TRUE;
          else{

if(priv_yearFrom1993==anotherTime.priv_yearFrom1993 &&
                  priv_monthInYear ==
anotherTime.priv_monthInYear &&
               priv_dayInMonth > anotherTime.priv_dayInMonth)
                  return TRUE;
              else{
                  if(priv_yearFrom1993 ==
                    anotherTime.priv_yearFrom1993 &&
                     priv_monthInYear ==
                    anotherTime.priv_monthInYear &&
```

```
                        priv_dayInMonth ==
                       anotherTime.priv_dayInMonth &&
                        priv_hourInDay >
anotherTime.priv_hourInDay)
                        return TRUE;
                      else{
                        if(priv_yearFrom1993 ==
                          anotherTime.priv_yearFrom1993 &&
                           priv_monthInYear ==
                          anotherTime.priv_monthInYear &&
                           priv_dayInMonth ==
                          anotherTime.priv_dayInMonth &&
                        priv_hourInDay ==
                       anotherTime.priv_hourInDay &&
                        priv_minuteInHour >
                               anotherTime.priv_minuteInHour)
                        return TRUE;
                        else
                            return FALSE;
                        }
                    }
                }
        }
}


char* Time::makeString( )
{
          char result[ 16 ];
          sprintf(result,"%.2d/%.2d/%.2d
%.2d:%.2d",priv_monthInYear, priv_dayInMonth,
              priv_yearFrom1993+93, priv_hourInDay ,
priv_minuteInHour );
          return (strdup (result));
}

void Time:: display()
{
          printf (" %d/%d/%d %d:%d  ", priv_monthInYear,
          priv_dayInMonth,priv_yearFrom1993, priv_hourInDay
,
                                          priv_minuteInHour );
}
```

My_String.h **********

```
#ifndef _My_String_H
#define _My_String_H

#include <iostream.h>


#include <stdio.h>
#include <string.h>

class My_String_rep
{
          char* str;
          int refs;
          int length; // does not include null byte
          My_String_rep(char *);
          My_String_rep(char **);
          friend class My_String;
};

class My_String
{
          My_String_rep *r;
          My_String(char**);
public:
          My_String(My_String&);
          My_String(char* = "");
          ~My_String();
          My_String& operator= (My_String);
          operator char *();
          const char* string();
          int Str_length();
          void print();
          friend int operator < (My_String, My_String);
          riend int operator > (My_String, My_String);
          friend int operator == (My_String, My_String);
          friend int operator != (My_String, My_String);
          friend My_String operator+ (My_String, My_String);
         friend My_String operator- (My_String, My_String);
          friend My_String operator- (My_String, int);
          friend ostream& operator<< (ostream&, My_String);
         friend istream& operator>> (istream&, My_String&);
};

#endif _My_String_H
```

My_String.cxx **********

```
#include "My_String.h"

My_String_rep::My_String_rep(char *s)
{
          str = new char[(length = strlen(s))  + 1];
          strcpy(str, s);
          refs = 1;
}

My_String_rep::My_String_rep(char** ptrptr)
{
          str = *ptrptr;
          refs = 1;
          length = strlen(str);
}

My_String::My_String(char *s)
{
          r = new My_String_rep(s);
}


My_String::My_String(char** ptrptr)
{
          r = new My_String_rep(ptrptr);
}

My_String::My_String(My_String &init)
{
          r = init.r;
          r ->refs++;
}

My_String& My_String::operator=(My_String str)
{
          if (!--r->refs)
          {
                    delete r->str;
                    delete r;
          }
          r = str.r;
          r -> refs++;
          return *this;
}
```

```
My_String::~My_String()
{
          if (!--r->refs)
          {
                    delete r->str;
                    delete r;
          }
}

My_String::operator char* ()
{
          char *p = new char[(r->length) + 1];
          strcpy(p,r->str);
          return p;
}

const char* My_String::string()
{
          return (r->str);
}


int operator< (My_String s1, My_String s2)
{
          return (strcmp(s1.string(), s2.string() )  < 0 );
}

int operator> (My_String s1, My_String s2)
{
          return ( strcmp(s1.string(), s2.string() ) > 0 );
}

int operator==(My_String s1, My_String s2)
{
          return ( strcmp ( s1.string(), s2.string() ) == 0 );
}

int operator!=(My_String s1, My_String s2)
{
          return( strcmp ( s1.string(), s2.string() ) != 0 );
}

ostream& operator<< (ostream& o, My_String s1)
{
          o << s1.string() ;
```

```
        return o;
}

istream& operator>> (istream& o, My_String& s2)
{
        char buf[256];
        o >> buf;
        s2 = My_String(buf);
        return o;
}


int My_String::Str_length()
{
        return (  r->length  );
}

void My_String::print()
{
        printf("%s", string() );
}

/* My_String operator+ (My_String s1, My_String s2)
{
    char *t;
    t = new char[s1.Str_length() + s2.Str_length() + 1];
    strcpy(t,s1.string());
    My_String s3(strcat(t,s2.string() ) );
    delete t;
    return s3;
}  */


My_String operator+ (My_String s1, My_String s2)
{
        char *t;
        t = new char[s1.Str_length() + s2.Str_length() +
1];
        strcpy(t,s1.string());
        strcat(t,s2.string());
        My_String s3(&t);
        return s3;
}
```

```
My_String operator- (My_String s1, My_String s2) {
          char* t;
          int n = s1.Str_length() - 2*s2.Str_length() - 4;
          t = new char[n + 1];
          char* p;
          strncpy(t, p = (char*)s1, n);
          t[n] = '\0';
          My_String s3(&t);
          delete[] p;
          return s3;
}

My_String operator- (My_String s1, int n) {
          char* t;
          int k = s1.Str_length()- n  ;
          t = new char[k+1];
          char* p;
          strncpy(t, p = (char*)s1, k);
          t[k] = '\0';
          My_String s3(&t);
          delete[] p;
          return s3;
}
```

C. PROGRAMS

There are two main Ada packages in addition to the main programs for the manager interface, designer interface and the Tae program on top of the manager interface. The two main packages are the Ada interface to C++ package that enables the Ada programs to access the data in the design database and the scheduler package that implements the scheduling algorithm. In the rest of this section we include a copy of the Ada code for both the specification and the implementation of both packages in addition to the code for the other main programs. The module dependency diagram for the relations between different program modules is shown in Figure 32.

1. The Ada Interface to DDB Package

ECS_Operations_s.a **********

```
-- Title              : The scheduler spec package
-- Author             : Salah badr
-- Date               : 4 Sept  1993
```

FIGURE 32. ECS Module dependency diagram

```
-- Revised            :
-- System             : Suns7
-- Compiler           : verdixAda
-- Description        :

     with TEXT_IO; -- BASIC_NUM_IO;
     use TEXT_IO; --, BASIC_NUM_IO ;
     with scheduler; use scheduler;

  --  generic

    package ECS_OPERATIONS is

package nat_io is new integer_io(natural);  use nat_io;

procedure system_call(command :string);


procedure auto_mail(name      : string;
                    step_id   : string);

procedure auto_mail2(name      : string;
                    step_id   : string);

procedure auto_mail3(name      : string;
                    step_id   : string);
--*************************************************************
--*                       COMPONENET OPERATIONS                *
--*************************************************************

procedure Create_Prototype(ddbname    :string;
                           option     :string;
                           protoname  :string);

procedure Show_Prototypes(ddbname    :string;
                          option     :string);

-- This function is general for all the operations: Show
-- Component,Show SubTree, Add New Version, Dump
-- Component,and Dump SubTree The only difference is the
-- option number

procedure general_function(ddbname    :string;
                           option     :string;
                           protoname  :string;
                           var_ver    :string;
```

```
                              comp_name  :string);

procedure Add_SubComponent(ddbname    :string;
                           option     :string;
                           protoname  :string;
                           var_ver    :string;
                           comp_name  :string;
                           var1_ver1  :string;
                           parent     :string);

procedure Dump_version(ddbname    :string;
                       option     :string;
                       protoname  :string;
                       comp_name  :string;
                       var1_ver1  :string);


--*************************************************************
--*            STEP OPERATIONS                                *
--*************************************************************

procedure Create_Step(ddbname    :string;
                      option     :string;
                      proto      :string;
                      comp       :string);

-- this function show either one step or a set of steps
-- according to the option given.
procedure Show_Step(ddbname    :string;
                    option     :string;
                    step_id    :string);

procedure get_sched_data(ddbname    :string;
                         step_id  :string);

procedure get_sched_data_2(ddbname    :string;
                           d_name     :string);

-- This function is general for all the operations: delete
-- input (primary, secondary or affected modules), updating
-- precedence,priority, exp_level duration, status or
-- designer's name. The only difference is the option number

procedure general_update(ddbname    :string;
                         option     :string;
                         step_id    :string;
```

```
                          value      :string);

procedure create_substep(ddbname    :string;
                         option     :string;
                         step_id    :string;
                         p_input    :string;
                               duration   :string);

-- This function does add step inputs primary, secondary or
-- affected modules. The only difference is the option number
procedure update_time(ddbname    :string;
                      option     :string;
                      step_id    :string;
                      the_time   :string);

procedure remove_step_from_schedule(ddbname    :string;
                                    option     :string;
                                    step_id    :string;
                                    theDate    :string);

procedure Early_Warning(ddbname    :string;
                        option     :string;
                        manager    :string);

--*************************************************************
--*            SCHEDULE OPERATIONS                            *
--*************************************************************
procedure Add_Assignment(ddbname     :string;
                         option      :string;
                         step_id     :string;
                         desname     :string;
                         the_time    :string;
                         start       :string;
                         finish      :string);

procedure get_current_time(the_time:out string);

procedure Show_Schedule(ddbname:string;
                        option :string);

procedure get_sched_data_1(ddbname    :string;
                           d_name     :string);

procedure Get_Schedule(ddbname:string;
                       option :string);
```

```
procedure Delete_Assignment(ddbname:string;
                            option :string;
                            step_id:string);

procedure Delete_Schedule(ddbname:string;
                          option :string);

procedure SAVE_SCHEDULE(HEAD : in LINK1;
                        dbname: string;
                        d_vector: in vector;
                         my_list : D_LINK );

procedure MAIN(indicator: in integer);
--*************************************************************
--*            DEASIGNER POOL OPERATIONS                      *
--*************************************************************
procedure DESIGNER_MENU;
procedure DESIGNER_OPERATIONS;


procedure Add_designer(ddbname:string;
                       option :string;
                       desname:string;
                       level  :string);

procedure Show_designer(ddbname:string;
                        option :string);

procedure put_designers(ddbname:string;
                        option :string);

procedure Delete_designer(ddbname:string;
                          option :string;
                          desname:string);

procedure Change_exp_level(ddbname:string;
                           option :string;
                           desname:string;
                           level  :string);

procedure Change_status(ddbname:string;
                        option :string;
                        desname:string);

end ECS_OPERATIONS;
```

ECS_Operations_b.a **********

```
-- Title                : Scheduler package body
-- Author               : Salah badr
-- Date                 : 4 Sept 1993
-- Revised              :
-- System               : Suns7
-- Compiler             : VerdixAda
-- Description          :

with text_io, system;
use text_io, system;
with scheduler; use scheduler;


package body ECS_OPERATIONS is
Name1           : STRING(1..9) := "supportDB";
option          : STRING(1..1) ;
Name3           : STRING(1..64);
Name4           : STRING(1..64);
Name5           : STRING(1..64);
Name6           : STRING(1..64);
Name7           : STRING(1..64);
D_status        : STRING(1..4);
SELECTOR        : natural := 0;
Length          : integer;
Length1          : integer;
data_file       : FILE_TYPE;
answer          : character :='y';
procedure system_call(command :string) is

        procedure system_C(command :address);
        pragma INTERFACE(C, system_C);
        pragma INTERFACE_NAME(system_C, "_system");
        temp : constant STRING := command&ASCII.NUL;
        error: integer;
begin
    system_C(TEMP'ADDRESS);
end system_call;
--*************************************************************

procedure auto_mail(name      : string;
                    step_id   : string)is
begin
   CREATE(data_file, OUT_FILE,"temp2");
   put(data_file,"You have been assigned the step no:   ");
```

```
   put(data_file,step_id);
   CLOSE(data_file);

   system_call("mail "&name&"< temp2");
   system_call("rm temp2");
end auto_mail;
--*************************************************************
procedure auto_mail1(name     : string;
                     step_id  : string)is
begin
   CREATE(data_file, OUT_FILE,"temp2");
   put(data_file,"ATTENTION REQUIRED  Step:  ");
   put(data_file,step_id);
   put(data_file,"   should commit within an hour...");
   CLOSE(data_file);
   system_call("mail "&name&"< temp2");
   system_call("rm temp2");
end auto_mail1;
--*************************************************************
procedure auto_mail2(name     : string;
                     step_id  : string)is
begin
   CREATE(data_file, OUT_FILE,"temp2");
   put(data_file,"Your current assigned step:  ");
   put(data_file,step_id);
   put(data_file,"   has been Suspended...");
   CLOSE(data_file);
   system_call("mail "&name&"< temp2");
   system_call("rm temp2");
end auto_mail2;
--*************************************************************
procedure auto_mail3(name     : string;
                  step_id  : string)is
begin
   CREATE(data_file, OUT_FILE,"temp2");
   put(data_file,"Your current assigned step:  ");
   put(data_file,step_id);
   put(data_file,"   has been abandoned...");
   CLOSE(data_file);
   system_call("mail "&name&"< temp2");
   system_call("rm temp2");
end auto_mail3;

--*************************************************************
--*      COMPONENT OPERATIONS                                 *
--*************************************************************
```

```
procedure Create_Prototype(ddbname    :string;
                           option     :string;
                           protoname  :string) is

begin
   system_call("maincomp "&" "&ddbname&" "&option&"
"&protoname);
end Create_Prototype;
--*************************************************************
procedure Show_Prototypes(ddbname    :string;
                          option     :string) is

begin
   system_call("maincomp "&" "&ddbname&" "&option&" >
ddbdisplay");
end Show_Prototypes;
--*************************************************************
-- This function is general for all the operations: Show
-- Component,Show SubTree, Add New Version, Dump Component,
-- and Dump SubTree The only difference is the option number

procedure general_function(ddbname    :string;
                           option     :string;
                           protoname  :string;
                           var_ver    :string;
                           comp_name  :string) is

begin
  system_call("maincomp "&ddbname&" "&option&" "&protoname&"
"&var_ver&" "&comp_name);
end general_function;
--*************************************************************
procedure Add_SubComponent(ddbname    :string;
                           option     :string;
                           protoname  :string;
                           var_ver    :string;
                           comp_name  :string;
                           var1_ver1  :string;
                           parent     :string) is

begin
   system_call("maincomp "&" "&ddbname&" "&option&"
"&protoname&" "
                &var_ver&" "&comp_name&" "&var1_ver1&"
"&parent);
end Add_SubComponent;
```

```
--*************************************************************
procedure Dump_version(ddbname    :string;
                       option     :string;
                       protoname  :string;
                       comp_name  :string;
                       var1_ver1  :string) is

begin
   system_call("maincomp "&" "&ddbname&" "&option&"
"&protoname&" "
                &comp_name&" "&var1_ver1);
end Dump_version;
--*************************************************************
--*          STEP OPERATIONS                                   *
--*************************************************************
-- option (1)
procedure Create_Step(ddbname    :string;
                      option     :string;
                      proto      :string;
                      comp       :string) is

begin
  system_call("mainstep "&" "&ddbname&" "&option&" "&proto&"
"&comp);
end Create_Step;
--*************************************************************
-- this function show either one step(2) or a set of steps(3)
-- according to the option given.Same function is used for
-- commit_step(e), approve(c), and commit_substep(d)

procedure Show_Step(ddbname    :string;
                    option     :string;
                    step_id    :string) is

begin
   system_call("mainstep "&" "&ddbname&" "&option&"
"&step_id&" > ddbdisplay");
end Show_Step;
--*************************************************************
-- This function is general for all the operations:
-- status(7) or designer's name(8).
-- The only difference is the option number

procedure general_update(ddbname    :string;
                         option     :string;
                         step_id  :string;
```

```
                          value    :string) is

begin
   system_call("mainstep "&" "&ddbname&" "&option&"
"&step_id&" "&value);
end general_update;
--*************************************************************
procedure create_substep(ddbname    :string;
                         option     :string;
                         step_id    :string;
                         p_input    :string;
                         duration   :string) is
          the_time:string(1..14);
begin
   get_current_time(the_time);
   system_call("mainstep "&" "&ddbname&" "&option&"
"&step_id&" "&p_input&" "
                &the_time&" "&duration&" >temp8");
end create_substep;
--*************************************************************
-- option (a)
procedure get_sched_data(ddbname    :string;
                         step_id  :string)is
        the_time:string(1..14);
begin
   get_current_time(the_time);
   system_call("mainstep "&" "&ddbname&" a "&step_id&"
"&the_time&" >temps");
end get_sched_data;
--*************************************************************
-- option (i)
procedure get_sched_data_2(ddbname    :string;
                           d_name     :string)is
        the_time:string(1..14);
begin
   get_current_time(the_time);
   system_call("mainstep "&" "&ddbname&" 1 "&the_time&"
"&d_name&" >temps");
end get_sched_data_2;
--*************************************************************
-- This function updates start_time(5), and finish_time(6)
procedure update_time(ddbname    :string;
                  option     :string;
                  step_id  :string;
                  the_time  :string) is
```

```
begin
   system_call("mainstep "&" "&ddbname&" "&option&"
"&step_id&" "
                &the_time);
end update_time;
--*************************************************************
-- option=f
procedure remove_step_from_schedule(ddbname    :string;
                                    option     :string;
                                           step_id
:string;
                                    theDate    :string) is
 begin
  system_call("mainstep "&" "&ddbname&" "&option&"
"&step_id&" "&theDate&" >temp4");
end remove_step_from_schedule;

--*************************************************************
-- option=k
procedure Early_Warning(ddbname    :string;
                        option     :string;
                        manager    :string) is
   the_time    :string(1..14);
   an_id       :string(1..5);
   Name4       :string(1..64);
   data_file1  : FILE_TYPE;
   begin
      get_current_time(the_time);
      system_call("mainstep "&" "&ddbname&" "&option&"
"&the_time&" >temp6");
      OPEN(data_file1, IN_FILE,"temp6");
      WHILE not END_OF_FILE(data_file1) loop
      get_line(data_file1,an_id,length);
      for i in Length+1..5 loop
         an_id(i):=' ';
       end loop;
       get_line(data_file1,Name4,length);
       for i in Length+1..64 loop
          Name4(i):=' ';
       end loop;
       auto_mail1(Name4,an_id);
       auto_mail1(manager,an_id);
       end loop;
       CLOSE(data_file1);
       system_call("rm temp6");
end Early_Warning;
```

```
--*************************************************************
--*            SCHEDULE OPERATIONS                            *
--*************************************************************

procedure Add_Assignment(ddbname     :string;
                         option      :string;
                         step_id     :string;
                         desname     :string;
                         the_time    :string;
                         start       :string;
                         finish      :string) is
begin
   system_call("mainsched "&" "&ddbname&" "&option&"
"&step_id&" "&desname&" "
               &the_time&" "&start&" "&finish);
end Add_Assignment;
--*************************************************************

procedure get_current_time(the_time:out string) is
begin
   system_call("date '+%m/%d/%y %H:%M' > temp1");
   OPEN(data_file, IN_FILE,"temp1");
   get(data_file,the_time);
   CLOSE(data_file);
   system_call("rm temp1");
end get_current_time;
--*************************************************************
-- option = 1
procedure Show_Schedule(ddbname:string;
                        option :string) is
begin
   system_call("mainsched "&" "&ddbname&" "&option&"
>ddbdisplay");
end Show_Schedule;
--*************************************************************
procedure get_sched_data_1(ddbname    :string;
                                 d_name     :string) is
        the_time:string(1..14);
begin
   get_current_time(the_time);
   system_call("mainstep "&" "&ddbname&" m "&the_time&"
"&d_name&" >temp3");
end get_sched_data_1;

--*************************************************************
-- option = 5
```

```
procedure Get_Schedule(ddbname:string;
                       option :string) is
begin
   system_call("mainsched "&" "&ddbname&" "&option);
end Get_Schedule;
--*************************************************************
--option =4
procedure Delete_Assignment(ddbname:string;
                            option :string;
                            step_id:string) is
begin
   system_call("mainsched "&" "&ddbname&" "&option&"
"&step_id);
end  Delete_Assignment;
--*************************************************************
-- option = 3
procedure Delete_Schedule(ddbname:string;
                          option :string) is
begin
   system_call("mainsched "&" "&ddbname&" "&option);
end Delete_Schedule;
--*************************************************************
--*           DESIGNER POOL OPERATIONS                        *
--*************************************************************
procedure Add_designer(ddbname:string;
                       option :string;
                       desname:string;
                       level  :string) is
begin
   system_call("maindes "&" "&ddbname&" "&option&"
"&desname&" "&level);
end Add_designer;
--*************************************************************
procedure Show_designer(ddbname:string;
                        option :string) is
begin
   system_call("maindes "&" "&ddbname&" "&option&" >
ddbdisplay");
end Show_designer;
--*************************************************************
procedure put_designers(ddbname:string;
                        option :string) is
begin
   system_call("maindes "&" "&ddbname&" "&option&" >temp2");
end put_designers;
--*************************************************************
```

```
procedure Delete_designer(ddbname:string;
                          option :string;
                          desname:string) is
begin
   system_call("maindes "&" "&ddbname&" 3 "&desname);
end  Delete_designer;
--*************************************************************

procedure Change_exp_level(ddbname:string;
                           option :string;
                           desname:string;
                           level  :string) is
begin
   system_call("maindes "&" "&ddbname&" 4 "&desname&"
"&level);
end Change_exp_level;
--*************************************************************

procedure Change_status(ddbname:string;
                        option :string;
                        desname:string) is
begin
   system_call("maindes "&" "&ddbname&" 5 "&desname);
end Change_status;

--*************************************************************

       -- DISPLAY THE MAIN MENU.
       procedure DESIGNER_MENU is
           begin
             new_line;
             set_col(25); put("MAIN MENU"); new_line;
             set_col(25); put("----------"); new_line(2);
             set_col(5); put("[1] Add designer");
             new_line;
             set_col(5); put("[2] Show designers");
             new_line;
             set_col(5); put("[3] Delete designer");
             new_line;
             set_col(5); put("[4] Change expertise level");
             new_line;
             set_col(5); put("[5] Return to main menu");
            new_line(3);
           set_col(5); put("Enter the number of your choice
:  ");
```

```
            end DESIGNER_MENU ;
--*************************************************************
procedure DESIGNER_OPERATIONS is

begin
          loop
           DESIGNER_MENU ;
              get(SELECTOR); skip_line ;
            case SELECTOR  is
               -- insert a record into database.
               when 1 =>
                    option:="1";
                  while answer = 'y' or answer='Y' loop
                    put("Enter designer's name: ");
                    get_line(Name3,Length);
                    for i in Length+1..64 loop
                        Name3(i):=' ';
                    end loop;
                    put("Enter Expertise Level: ");
                    get_line(Name4,Length);
                    for i in Length+1..64 loop
                        Name4(i):=' ';
                    end loop;
                    Add_designer(Name1,option,Name3,Name4);
                   put("Add more designers [Answer y/n]: ");
                    get(answer); skip_line ;
                  end loop;
                   put_designers(Name1,"2");
                   get_sched_data(Name1,"0");
                   get_sched_data_1(Name1,Name3);
                   main(0);
               when 2 =>    -- Show designers
                    option:="2";
                    Show_designer(Name1,option);
               when 3 =>    -- Delete designer
                    option:="3";
                    put("Enter designer's name: ");
                    get_line(Name3,Length);
                    for i in Length+1..64 loop
                        Name3(i):=' ';
                    end loop;
                    Show_designer(Name1,"2");
                    OPEN(data_file, IN_FILE,"ddbdisplay");
                    WHILE not END_OF_FILE(data_file) loop
                    get_line(data_file,Name4,length1);
```

```
            for i in Length1+1..64 loop
            Name4(i):=' ';
            end loop;
            put_line(Name4(1..length));
            put_line(Name4(45..48));
            if Name4(1..length)=Name3(1..length)
            then D_status:=Name4(45..48);
            end if;
            end loop;
            CLOSE(data_file);
            system_call("rm ddbdisplay");
            if D_status(1..4)= "Busy" then
            put_line("Desiner is busy, Confirmation
                                Required");
            put("Do you have to delete him now(answer
                                y/n): ");
            get(answer); skip_line;
            if answer ='y' or answer = 'Y' then
            Delete_designer(Name1,option,Name3);
            system_call("mainsched "&Name1&" 7
                                "&Name3);
            put_designers(Name1,"2");
            get_sched_data(Name1,"0");
            get_sched_data_1(Name1,Name3);
            main(0);
            end if;
            else
                Delete_designer(Name1,option,Name3);
             end if;
        when 4 =>       -- Change expertise level
            option:="4";
            put("Enter designer's name: ");
            get_line(Name3,Length);
            for i in Length+1..64 loop
                Name3(i):=' ';
            end loop;
            put("Enter Expertise Level: ");
            get_line(Name4,Length);
            for i in Length+1..64 loop
                Name4(i):=' ';
            end loop;
          Change_exp_level(Name1,option,Name3,Name4);
            put_designers(Name1,"2");
            get_sched_data(Name1,"0");
            get_sched_data_1(Name1,Name3);
            main(0);
```

```
                when 5 =>
--                  put("thank you .... Bye ...Bye");
                   new_line ;
                  exit;
                -- exception handling for selector case.
                when others =>
                  put(" BAD CHOICE. PLEASE TRY AGAIN");
                  new_line ;
            end case;
          end loop;
end DESIGNER_OPERATIONS;

--*************************************************************
procedure SAVE_SCHEDULE (HEAD    : in LINK1;
                         dbname  : string;
                         d_vector: in vector;
                         my_list : D_LINK ) is
        CURRENT : LINK1 := HEAD ;
        my_time:string(1..14);
        an_id   :string(1..3);
        start   :string(1..3);
        finish  :string(1..3);
--        designer :string(1..64);
        my_name :string(1..16);
        my_index: natural;
--        length  : natural;
    begin
      get_current_time(my_time);
      while CURRENT /= null loop
         put(an_id,CURRENT.STEP_ID);
         put(start,CURRENT.START_TIME);
         put(finish, CURRENT.FINISH_TIME);
         if CURRENT.DESIGNER_LEVEL = high then
            my_index := CURRENT.DESIGNER_NO +
                                  d_vector(1)+d_vector(2);
         elsif CURRENT.DESIGNER_LEVEL = medium then
               my_index := CURRENT.DESIGNER_NO + d_vector(1);
         else
               my_index := CURRENT.DESIGNER_NO ;
         end if;
         FIND_DESIGNER(my_list, my_index, my_name);
         if CURRENT.START_TIME = 0
         then auto_mail(my_name, an_id);
              Change_status(Name1,"5", my_name);
              general_update(Name1,"7",an_id,"3");
              general_update(Name1,"5",an_id,my_time);
```

```
          else
                general_update(Name1,"7",an_id,"2");
          end if;
          general_update(Name1,"8",an_id,my_name);
          Add_Assignment(dbname,"1",an_id, my_name,
                 my_time,start, finish);
         CURRENT := CURRENT.NEXT ;
       end loop;
    end SAVE_SCHEDULE;
--************************************************************
    procedure MAIN(indicator :in integer) is

      an_id                :string(1..3);
      STEP_LIST            : LINK := null  ;
      STEP_LIST_TAIL       : LINK := null  ;
      POSITION             : LINK ;
      PREVIOUS             : LINK := null ;
      CURRENT              : LINK ;
      temp1                : LINK;
      SCHEDULE_TAIL        : LINK1 := null;
      T, k,  temp          : natural;
      S_ID          : natural; -- step_id
      R             : natural:= 1; -- row
      C             : natural:= 1; -- column
      FOUND           :boolean := FALSE;
      FEASIBLE        :boolean := TRUE;
      my_SELECTOR        : natural := 0;
      LEVELS          : natural := 3 ;
      READY_LIST: LIST_VECTOR(1..LEVELS):= (others => null);
      SCHEDULE  : SCHEDULE_VECTOR(1..5):= (others => null);
      PENDING_LIST: link := null;
      ASSIGNED : boolean := TRUE;
      answer   :character :='n';
      level_l, level_m, Level_h, INDEX  : natural;
      DES_LIST : D_LINK;
--************************************************************
begin
      begin
          CREATE_DESIGNER_LIST(DES_LIST,level_l, level_m,
                               Level_h);
        if level_l >= level_m and level_l >= Level_h then
                               INDEX:= level_l;
       elsif level_m >= level_l and level_m >= Level_h then
                               INDEX:= level_m;
        else INDEX:= level_h;
        end if;
```

```
end;

DECLARE
designer_vector : VECTOR(1..3) := (level_l, level_m,
                        Level_h);
EAT : designer_matrix(1..LEVELS, 1..INDEX) := (others
                        => (others => 0));


-- Main program.
begin

      -- insert a record into database.
     CREATE_AND_INSERT_NEW_STEP(STEP_LIST,
                                    STEP_LIST_TAIL);
   if STEP_LIST /= null then
   RESET_FOR_RESCHEDULING (DES_LIST,STEP_LIST, EAT,
                                    designer_vector);
   k := list_size(STEP_LIST);
   CURRENT := STEP_LIST;
  while CURRENT /= null loop   -- initialize ready_lists
         if CURRENT.IN_DEGREE = 0 then
         temp:=EXPERTISE_LEVEL'POS(CURRENT.EXP_LEVEL)+1;
         INSERT_ORDER_START_TIME(READY_LIST(temp),
                                    CURRENT);
         CURRENT  := CURRENT.NEXT;
         else
             CURRENT  := CURRENT.NEXT;
         end if;
   end loop;
   while k /= 0 loop
    for i in reverse  1..LEVELS loop
    ASSIGNED := TRUE;
    -- Schedule the steps
     while READY_LIST(i) /= null and FEASIBLE and
                        ASSIGNED loop
    STRONGLY_FEASIBLE(READY_LIST(i), EAT,
              DESIGNER_VECTOR, LEVELS,  FEASIBLE);
    if FEASIBLE
    then
     temp1 := READY_LIST(i);
     LEVEL_MINMUM(EAT, temp1.EXP_LEVEL,LEVELS,
                        DESIGNER_VECTOR, R,C);
     ASSIGN_STEP(READY_LIST,STEP_LIST,EAT,
         DESIGNER_VECTOR,R,C,i,SCHEDULE(2), ASSIGNED);
       if ASSIGNED then
```

```
                k := k - 1;
              CHECK_IN_DEGREE_1(temp1, STEP_LIST,READY_LIST,
                                                  EAT(R,C));
              end if;
            else exit;
            end if;
            end loop;
            if not feasible then exit; end if;
            end loop;
            if not feasible then exit; end if;
            end loop;
            SCHEDULE_RECORD_HEADING ;
            PRINT_ALL_SCHEDULE_RECORDS (SCHEDULE(2),DES_LIST,
                                        designer_vector);
            if indicator = 1 then
           put("Confirmation required to save the schedule in
                           DDB. Answer(Y/N): ");
                  get(answer);
                  if answer ='y' or answer='Y'
                  then
                    Save_Schedule(SCHEDULE(2),NAME1,
                           DESIGNER_VECTOR, DES_LIST);
                  end if;
               else
                 Save_Schedule(SCHEDULE(2),NAME1,
                           DESIGNER_VECTOR, DES_LIST);
               end if;
               POSITION := STEP_LIST;
               While POSITION /= null loop
                     if POSITION.DEADLINE_CHANGE then
                        put(an_id,POSITION.STEP_ID);
                        system_call("mainstep "&Name1&" j
                              "&an_id);
                     end if;
                     POSITION := POSITION.NEXT;
               end loop;
            end if;
     end;
     end MAIN;
end ECS_OPERATIONS;
```

2. The Scheduler Package

scheduler_s.a **********

```
-- Title              : The scheduler spec package
```

```
-- Author              : Salah badr
-- Date                : 25 May  1993
-- Revised             :
-- System              : Suns7
-- Compiler            : VerdixAda
-- Description         :

     with generic_set_pkg;
     with generic_map_pkg;
     with TEXT_IO; -- BASIC_NUM_IO;
     use TEXT_IO; --, BASIC_NUM_IO ;
     with test_io_pkg; use test_io_pkg;

  --  generic

package scheduler is

   package nat_set is new generic_set_pkg(natural, 5); use
nat_set;
   -- instantiate instances of the generic map package.
   package nat_map is
               new generic_map_pkg(key => natural, result =>
natural);
   package set_map is
          new generic_map_pkg(key => natural, result => set) ;



   type EXPERTISE_LEVEL is (low, medium, high);
   package exp_map is
    new generic_map_pkg(key => natural, result =>
EXPERTISE_LEVEL);


     type STEP_RECORD ;  -- is limited private;
     type LINK is access STEP_RECORD;

     type DESIGNER_RECORD;  -- is limited private;
     type D_LINK is access DESIGNER_RECORD;

     type SCHEDULE_RECORD;  -- is limited private;
     type LINK1 is access SCHEDULE_RECORD;

     type STEP_RECORD is record
               NEXT ,
               NEXT_READY,
```

```
            NEXT_PENDING        : LINK;
            STEP_ID             : natural;
            DEADLINE            : natural := 0;
            PRIORITY            : natural;
            ESTIMATED_DURATION  : natural;
            EARLIEST_START_TIME: natural :=0;
            EXP_LEVEL           : EXPERTISE_LEVEL;
            SUCCESSORS          :  set;
            PREDECESSORS        :  set;
            IN_DEGREE           : natural;
            DEADLINE_CHANGE     : BOOLEAN := FALSE;
        end record ;
--*************************************************************
      type DESIGNER_RECORD is record
            NEXT                : D_LINK;
              D_name              : string(1..16);
            LEVEL    : EXPERTISE_LEVEL;
        end record ;
--*************************************************************

    type SCHEDULE_RECORD is record
            NEXT                : LINK1;
            STEP_ID             : natural;
            START_TIME          : natural;
            FINISH_TIME         : natural;
            DESIGNER_NO         : natural;
              STEP_LEVEL          : EXPERTISE_LEVEL;
            DESIGNER_LEVEL      : EXPERTISE_LEVEL;
      end record ;

        type VECTOR is array (POSITIVE range <>) of integer;
        type designer_matrix is array (POSITIVE range <>,
                              POSITIVE range <>)of natural;
        type LIST_VECTOR is array (POSITIVE range <>) of
                                        link;
        type SCHEDULE_VECTOR is array (POSITIVE range <>)
                                         of link1;
        DEADLINE1           : nat_map.map;
        PRIORITY1           : nat_map.map;
        ESTIMATED_DURATION1 : nat_map.map;
        EARLIEST_START_TIME1: nat_map.map;
        EXP_LEVEL1          : exp_map.map;
        SUCCESSORS1         :  set_map.map;
        PREDECESSORS1       :  set_map.map;

--*************************************************************
```

```
        -- Creating new step.
        procedure CREATE_NEW_STEP;

        -- Linking a step to the tail of the step list
        procedure INSERT_NEW_STEP (LIST ,TAIL,
                                   A_RECORD : in out LINK );

        -- Creating new step from a file and linking it to
        -- step list.
        procedure CREATE_AND_INSERT_NEW_STEP (LIST , TAIL :
                                              in out   LINK );

    procedure CREATE_DESIGNER_LIST(D_LIST : in out D_LINK;
                                   NO_LOW : in out natural;
                                   NO_MED : in out natural;
                                   NO_HIG : in out natural);

    procedure INSERT_D_record(HEAD,A_RECORD:in out D_LINK
);

     procedure FIND_DESIGNER(HEAD    : in D_LINK;
                             D_INDEX : in natural;
                             d_name  : out string);

        procedure set_successors(LIST : in out LINK);

        -- resetting the in_degree of the steps in the step
        -- list for rescheduling
        procedure RESET_IN_DEGREE(LIST : in out LINK);

        -- creating a new schedule record
        procedure CREATE_SCHEDULE_RECORD(
                          S_ID        : in natural;
                             TIME1    : in natural;
                             TIME      : in natural;
                             D_NO    : in natural;
                             S_LEVEL : in EXPERTISE_LEVEL;
                             D_LEVEL  : in EXPERTISE_LEVEL);


--************************************************************
        -- DISPLAY THE MAIN MENU.
        procedure SCHEDULER_MENU;

        -- PRINTING A SCHEDULE  HEADING LINE BEFORE PRINTING
        -- ANY RECORD.
```

```
        procedure SCHEDULE_RECORD_HEADING;

        -- display a record given its position in the list.
        procedure DISPLAY_SCHEDULE_RECORD(
                              CURRENT:in LINK1;
                               my_list : D_LINK ;
                               d_vector: in vector );

        -- print all the records in the SCHEDULE list.
         procedure PRINT_ALL_SCHEDULE_RECORDS(
                                        HEAD : in LINK1;
                                        my_list:D_LINK ;
                                      d_vector: in vector );

        -- PRINTING A STEP  HEADING LINE BEFORE PRINTING ANY
        -- RECORD.
        procedure STEP_RECORD_HEADING;

        -- display a record given its position in the list.
        procedure DISPLAY_STEP_RECORD (CURRENT : in LINK );

        -- print all the records in the STEP list.
         procedure PRINT_ALL_STEP_RECORDS(HEAD : in LINK );

        -- print all the records in the READY QUEUE.
        procedure PRINT_ALL_QUEUE_RECORDS(HEAD : in LINK );
--************************************************************
        -- Linking a step to the ready list in order of its
        -- deadline.
        procedure INSERT_ORDER_DEADLINE(
                             R_QUEUE :in out LINK;
                              A_RECORD  : in LINK   );

        -- Linking a step to the ready list in order of its
        -- start time.
        procedure INSERT_ORDER_START_TIME(R_QUEUE ,
                                  A_RECORD : in out LINK );

        -- Linking a step to the pending list in order of
        -- its start time.
        procedure INSERT_PENDING_ORDER_START_TIME(R_QUEUE,
                                  A_RECORD : in out LINK );

        -- Linking a step to the ready list in order of its
        -- (DEADLINE + start time).
        procedure INSERT_ORDER_MIXED(R_QUEUE ,
```

```
                                    A_RECORD : in out LINK );

        -- Linking a step to the ready list in order of its
        -- laxity.
        procedure INSERT_ORDER_LAXITY(R_QUEUE ,
                                    A_RECORD : in out LINK );
        -- Linking a schedule record to the tail of the
        -- schedule list
        procedure INSERT_NEW_SCHEDULE_RECORD(LIST ,TAIL,
                                  A_RECORD : in out LINK1);

        -- Linking a schedule record according to its
        -- expertise level
        procedure INSERT_ORDER_EXP_LEVEL(HEAD, A_RECORD:
                                       in out LINK1 );

        -- Linking a schedule_record to the schedule in
        -- order of
        -- its start time.
        procedure INSERT_ORDER_START_TIME (HEAD, A_RECORD :
                                       in out LINK1 );

        -- Linking a schedule_record to the schedule in
        -- order of its step id.
        procedure INSERT_ORDER_STEP_ID (HEAD, A_RECORD : in
                                       out LINK1 );
--************************************************************
        procedure LEVEL_MINMUM(MATRIX : in DESIGNER_MATRIX;
                               LEVEL    : in EXPERTISE_LEVEL;
                               MAX_LEVEL  : in natural;
                               ROW_LENGTH : in vector ;
                               L, J       : in out natural);

        function ROW_MINMUM(MATRIX     : in DESIGNER_MATRIX;
                            LEVEL      : in EXPERTISE_LEVEL;
                            ROW_LENGTH : in vector)return
                                                 natural ;

--************************************************************
        -- Search for target step. Return the position and
        -- previous if found
        -- " Record not found " if not found.
        procedure FIND_STEP (HEAD     : in     LINK;
                             S_ID     : in  natural;
                             POSITION   : in out LINK;
                             PREVIOUS   : in out LINK;
```

```
                                FOUND      : in out boolean);

        -- Delete a  step from step list.
        procedure DELETE_FROM_STEP_LIST (HEAD: in out LINK;
                                         POSITION : in LINK;
                                        PREVIOUS : in LINK);

      -- Delete a  step from pending list
      procedure DELETE_FROM_PENDING_LIST (HEAD : in out LINK;
                                          POSITION : in LINK;
                                         PREVIOUS : in LINK);

      -- Delete a  step from schedule.
      procedure DELETE_FROM_SCHEDULE (HEAD     : in out LINK1;
                                      POSITION : in LINK1;
                                     PREVIOUS : in LINK1);

        -- Delete a  step found by search and relink step
        -- list.
        procedure DELETE_FROM_READY_QUEUE ( HEAD: in out
                                                 LINK);

--*************************************************************
-- Decrementing the in_degree of the successors of a step
      procedure DECREMENT_IN_DEGREE (STEP, LIST: in out LINK;
                                     finish_t   : in natural);

        function list_size(LIST :in LINK) return natural;

       -- checking the in_degree of the successors of the
        -- assigned step
        -- This works with deadline heuristic (mixed and
        -- laxity too)
        procedure CHECK_IN_DEGREE (
                   STEP, R_QUEUE , LIST  : in out LINK;
                   finish_t              : in natural);

        -- checking the in_degree of the successors of the
        -- assigned step
        -- This works with start time heuristic.
      procedure CHECK_IN_DEGREE_1 (STEP, LIST  : in out LINK;
                             R_QUEUE     : in out LIST_VECTOR;
                               finish_t    : in natural);

      procedure CHECK_IN_DEGREE_2 (STEP, LIST  : in out LINK;
                             R_QUEUE      : in out LIST_VECTOR;
```

```
                                finish_t       : in natural);

          -- checking the pending list for ready steps of a
          -- certainlevel and insert them into the
          -- corresponding ready_list according to their
          -- deadlines
          procedure GET_READY_STEPS(t, k       : in natural ;
                                    LIST : in out LINK;
                                    R_QUEUE : in out LIST_VECTOR);
           -- checking the pending list for ready steps of a
          -- certain level and insert them into the
          -- corresponding ready_list
          -- according to their (deadlines + start times).
          procedure GET_READY_STEPS_1(t, k      : in natural ;
                                LIST, R_QUEUE     : in out LINK);

          -- checking the pending list for ready steps of a
          -- certain level and insert them into the
          -- corresponding ready_list according to their
          -- LAXITY.
          procedure GET_READY_STEPS_2(t, k : in natural ;
                                LIST, R_QUEUE     : in out LINK);

        -- get the top steps in the ready list and
        -- insert them into the corresponding ready_list
        -- according to their deadlines
          procedure GET_READY_STEPS (LIST :in out LINK;
                                R_QUEUE      : in out LIST_VECTOR);

        -- get the top steps in the ready list and
        -- insert them into the corresponding ready_list
        -- according to their (deadlines + start times).
          procedure GET_READY_STEPS_1 (LIST     :in out LINK;
                                R_QUEUE  : in out LIST_VECTOR);

        -- get the top steps in the ready list and
        -- insert them into the corresponding ready_list
        -- according to their Laxity.
          procedure GET_READY_STEPS_2 (LIST :in out LINK;
                                R_QUEUE   : in out LIST_VECTOR);
--*********************************************************
          procedure SUGGEST_DEADLINE_SLIP(step : in out link;
                                          value :in natural);

          -- checking the feasibility of the schedule with
          -- each step in the ready queue
```

```
        procedure STRONGLY_FEASIBLE(R_QUEUE     : in LINK;
                                    MATRIX      : in DESIGNER_MATRIX;
                                     D_VECTOR       : in vector;
                                     MAX_LEVEL      : in natural;
                                    FEASIBLE       : in out boolean );
--*************************************************************

      -- Assign a step to a designer according to its deadline
       -- and its expertise level
       procedure ASSIGN_STEP(R_QUEUE    : in out LIST_VECTOR;
                             LIST       : in out  LINK;
                            MATRIX    : in out DESIGNER_MATRIX;
                             ROW_LENGTH     : in vector;
                             M,N        : in out natural;
                             L              : in natural;
                             SCH            : in out LINK1;
                             done           : out boolean );


         -- reset the step list and the schedule for
         -- incrementing the schedule
         -- with new steps at certain time
         procedure RESET_FOR_RESCHEDULING(
                   d_list :in out D_LINK;
                    LIST  : in out LINK;
                   MATRIX       : in out DESIGNER_MATRIX;
                    d_vector    : in vector);

       procedure FIND_DESIGNER_POSITION(HEAD    : in D_LINK;
                                        D_INDEX : out natural;
                                         d_name  : in string);
-- *************************************************************

end scheduler;
```

scheduler_b.a **********

```
-- Title              : Scheduler package body
-- Author             : Salah badr
-- Date               : 25 May 1993
-- Revised            :
-- System             : Suns7
-- Compiler           : VerdixAda
-- Description        :

    with UNCHECKED_DEALLOCATION;
```

```
    package body scheduler is


    package nat_io is new integer_io(natural); use nat_io;
          procedure put_set is new generic_put;
          procedure get_set is new generic_input;
          procedure getf_set is new generic_file_input;
          procedure FREE is new
          UNCHECKED_DEALLOCATION(STEP_RECORD, LINK);
          procedure FREE1 is new
          UNCHECKED_DEALLOCATION(SCHEDULE_RECORD, LINK1);
        package enu_io is new
                    text_io.ENUMERATION_IO(EXPERTISE_LEVEL);

          STEP_ID          : natural :=1;
          NEW_RECORD       : LINK;-- new step record
          NEW_S_RECORD     : LINK1;-- new schedule record
          MY_RECORD        : D_LINK;-- new designer record
          -- logical file definitions
          data_file,data_file1   : FILE_TYPE;
          input            : string(1..10);
          -- physical file name that include step data
          E_level_error    :exception;
--*************************************************************
       -- Creating new step for standard input.
       procedure CREATE_NEW_STEP  is
           begin
              NEW_RECORD := new STEP_RECORD ;
              -- assign values to record fields.
              NEW_RECORD.STEP_ID := STEP_ID;
              put_line("Please Enter  DEADLINE ");
              get(NEW_RECORD.DEADLINE );
              put_line("Please Enter  PRIORITY ");
              get(NEW_RECORD.PRIORITY );
              put_line("Please Enter  ESTIMATED_DURATION ");
              get(NEW_RECORD.ESTIMATED_DURATION );
             put_line("Please Enter  EARLIEST START TIME  ");
              get(NEW_RECORD.EARLIEST_START_TIME);
              put_line("Please Enter  PREDECESSORS  ");
              get_set(NEW_RECORD.PREDECESSORS);
              put_line("Please Enter  SUCCESSORS  ");
              get_set(NEW_RECORD.SUCCESSORS);
           put_line("Please Enter  EXPERTISE LEVEL REQUIRED
");
              enu_io.get(NEW_RECORD.EXP_LEVEL);
```

```
--                PREDECESSORS := NEW_RECORD.PREDECESSORS;
                  NEW_RECORD.IN_DEGREE :=
size(NEW_RECORD.PREDECESSORS);
               skip_line;
            end CREATE_NEW_STEP;
--  ********************************************************
            -- Linking a step to the tail of the step list
          procedure INSERT_NEW_STEP (LIST , TAIL,
                                  A_RECORD : in out   LINK ) is

              begin
                     if LIST = null
               then A_RECORD.NEXT := LIST;
                    LIST := A_RECORD ;
                     else
                       TAIL.NEXT := A_RECORD ;
                     end if;
               TAIL := A_RECORD ;
              end INSERT_NEW_STEP;

--  ********************************************************
       -- Creating new step from a file.
       procedure CREATE_AND_INSERT_NEW_STEP(LIST, TAIL:
                                          in out   LINK )  is
           begin
              OPEN(data_file, IN_FILE, "temps");
              WHILE not END_OF_FILE(data_file) loop
              NEW_RECORD := new STEP_RECORD ;
              -- assign values to record fields.
              get(data_file,NEW_RECORD.STEP_ID );
              get(data_file,NEW_RECORD.DEADLINE );
              get(data_file, NEW_RECORD.PRIORITY );
              get(data_file, NEW_RECORD.ESTIMATED_DURATION );
              getf_set(data_file, NEW_RECORD.PREDECESSORS);
              enu_io.get(data_file, NEW_RECORD.EXP_LEVEL);
              get(data_file,NEW_RECORD.IN_DEGREE);
              INSERT_NEW_STEP(LIST, TAIL,NEW_RECORD);
                end loop;
                CLOSE (data_file);
              set_successors(LIST);
           exception
              when DATA_ERROR =>
                  put_line("Step is not in approved state");
              when constraint_error =>
                  put_line("Step is not in approved state");
              when others =>
```

```
                    null;
            end CREATE_AND_INSERT_NEW_STEP;

--*************************************************************
          procedure CREATE_DESIGNER_LIST(D_LIST : in out
                                                  D_LINK;
                                      NO_LOW : in out natural;
                                      NO_MED : in out natural;
                                     NO_HIG : in out natural)is
--           Length : integer;
           begin
              NO_LOW := 0;
              NO_MED := 0;
              NO_HIG := 0;
              OPEN(data_file1, IN_FILE, "temp2");
              WHILE not END_OF_FILE(data_file1) loop
                MY_RECORD := new DESIGNER_RECORD ;
                -- assign values to record fields.
                get(data_file1,MY_RECORD.D_name);
--                for i in Length+1..20 loop
--                        MY_RECORD.D_name(i):=' ';
--                end loop;
--                put(MY_RECORD.D_name);
                enu_io.get(data_file1,MY_RECORD.LEVEL );
                skip_line(data_file1);
--                enu_io.put(MY_RECORD.LEVEL);
--                new_line;
             if MY_RECORD.LEVEL = low then NO_LOW:=NO_LOW +1;
              elsif MY_RECORD.LEVEL = medium then NO_MED :=
                                 NO_MED +1;
                else   NO_HIG := NO_HIG + 1;
                end if;
                    insert_D_record(D_LIST,MY_RECORD);
              end loop;
              CLOSE (data_file1);
          end CREATE_DESIGNER_LIST;
--*************************************************************
          -- Linking a deigner record according to its
          -- expertise level
        procedure INSERT_D_record (HEAD , A_RECORD : in out
                                            D_LINK ) is
            CURRENT  : D_LINK := HEAD ;
            PREVIOUS : D_LINK := null;
            LARGER_FOUND : BOOLEAN := FALSE ;
            begin
             while CURRENT /= null and not LARGER_FOUND loop
```

```
            if CURRENT.LEVEL > A_RECORD.LEVEL then
               LARGER_FOUND := TRUE  ;
            else
               PREVIOUS := CURRENT ;
               CURRENT  := CURRENT.NEXT ;
            end if ;
            end loop;
            A_RECORD.NEXT := CURRENT ;
            if PREVIOUS = null then
               HEAD     := A_RECORD ;
            else
               PREVIOUS.NEXT := A_RECORD ;
            end if ;
          end INSERT_D_record;
--*************************************************************
          -- Search for target step. Return the position and
          -- previous if found,
          -- " Record not found " if not found.
        procedure FIND_DESIGNER(HEAD    : in D_LINK;
                                D_INDEX : in natural;
                                d_name  : out string) is
            POSITION  : D_LINK := HEAD ;
            K : natural :=1;
         begin
            while POSITION /= null and K /= D_INDEX loop
               POSITION := POSITION.NEXT ;
               k := k+1;
            end loop ;
              d_name := POSITION.D_name;
          end FIND_DESIGNER ;

--*************************************************************
          procedure set_successors(LIST : in out LINK) is
            CURRENT : LINK := LIST;
            current1: LINK;
             begin
               while CURRENT /= null loop
                      if size(CURRENT.PREDECESSORS) /= 0
                  then
                             current1:=LIST;
                       while current1 /= null loop
                          if member(current1.step_id,
                             CURRENT.PREDECESSORS)
                          then add(CURRENT.STEP_ID,
                             current1.successors);
                          end if;
```

```
                                current1 := current1.next;
                            end loop;
                        end if;
                        CURRENT := CURRENT.NEXT;
                    end loop;
            end set_successors;
--*************************************************************
        -- resetting the in_degree of the steps in the step list
          -- for rescheduling
          procedure RESET_IN_DEGREE(LIST : in out LINK) is
                CURRENT : LINK := LIST;
                 begin
                      while CURRENT /= null loop
                            CURRENT.IN_DEGREE :=
                                  size(CURRENT.PREDECESSORS);
                            CURRENT := CURRENT.NEXT;
                      end loop;
                 end RESET_IN_DEGREE;

--*************************************************************
            -- creating a new schedule record
            procedure CREATE_SCHEDULE_RECORD (
                                  S_ID : in natural;
                                  TIME1 : in natural;
                                  TIME2 : in natural;
                                  D_NO : in natural;
                                  S_LEVEL: in EXPERTISE_LEVEL;
                                  D_LEVEL:in EXPERTISE_LEVEL) is
               begin
                      NEW_S_RECORD := new SCHEDULE_RECORD ;
                      NEW_S_RECORD.STEP_ID := S_ID;
                      NEW_S_RECORD.START_TIME := TIME1;
                      NEW_S_RECORD.FINISH_TIME := TIME2;
                      NEW_S_RECORD.DESIGNER_NO := D_NO;
                      NEW_S_RECORD.STEP_LEVEL :=S_LEVEL;
                      NEW_S_RECORD.DESIGNER_LEVEL := D_LEVEL;
               end  CREATE_SCHEDULE_RECORD;

--*************************************************************
            -- PRINTING A SCHEDULE  HEADING LINE BEFORE PRINTING
            -- ANY RECORD.
            procedure SCHEDULE_RECORD_HEADING  is
               begin
                      put("STEP_ID   S_LEVEL  D_NAME
START_TIME  FINISH_TIME");
                      new_line;
```

```
                put("-------   -------  ----------  ----
------  -----------");
            new_line;
        end SCHEDULE_RECORD_HEADING ;

        -- display a record given its position in the list.
        procedure DISPLAY_SCHEDULE_RECORD(
                              CURRENT : in LINK1;
                               my_list : D_LINK;
                               d_vector: in vector ) is
        my_index : natural;
        my_name  : string(1..16);
        begin
                SET_COL(4);
             test_io_pkg.put(CURRENT.STEP_ID);
                SET_COL(11);
                enu_io.put(CURRENT.STEP_LEVEL);
                SET_COL(21);
             if CURRENT.DESIGNER_LEVEL = high then
               my_index := CURRENT.DESIGNER_NO +
                           d_vector(1)+d_vector(2);
              elsif CURRENT.DESIGNER_LEVEL = medium then
            my_index := CURRENT.DESIGNER_NO + d_vector(1);
              else
                my_index := CURRENT.DESIGNER_NO ;
              end if;
              FIND_DESIGNER(my_list, my_index, my_name);
                 put(my_name);
                 SET_COL(37);
                 test_io_pkg.put(CURRENT.START_TIME);
                 SET_COL(48);
                 test_io_pkg.put(CURRENT.FINISH_TIME);
                 new_line;
        end DISPLAY_SCHEDULE_RECORD;

        -- print all the records in the SCHEDULE list.
        procedure PRINT_ALL_SCHEDULE_RECORDS (
                             HEAD    : in LINK1;
                             my_list : D_LINK;
                             d_vector: in vector ) is
           CURRENT : LINK1 := HEAD ;
        begin
           while CURRENT /= null loop
               DISPLAY_SCHEDULE_RECORD (CURRENT,my_list,
                   d_vector) ;
                 CURRENT := CURRENT.NEXT ;
```

```
            end loop;
          end PRINT_ALL_SCHEDULE_RECORDS ;
--*************************************************************
          -- PRINTING A STEP  HEADING LINE BEFORE PRINTING ANY
          -- RECORD.
          procedure STEP_RECORD_HEADING  is
          begin
          put("STEP_ID  DEADLINE  PRIORITY  PREDECE  SUCCESS
E_LEVEL  IN_DEGREE");
          new_line;
          put("------   -------   --------  ------    ------
-  -------  --------");
          new_line;
           end STEP_RECORD_HEADING ;

          -- display a record given its position in the list.
          procedure DISPLAY_STEP_RECORD(CURRENT :in LINK ) is
             temp1 : natural;
           begin
                    SET_COL(4);
                    test_io_pkg.put(CURRENT.STEP_ID);
                    SET_COL(12);
                    test_io_pkg.put(CURRENT.DEADLINE);
                    SET_COL(23);
                    test_io_pkg.put(CURRENT.PRIORITY);
                    SET_COL(31);
                    put_set(CURRENT.PREDECESSORS);
                    SET_COL(41);
                    put_set(CURRENT.SUCCESSORS);
                    SET_COL(49);
                    enu_io.put(CURRENT.EXP_LEVEL);
                    SET_COL(61);
                    test_io_pkg.put(CURRENT.IN_DEGREE);
                    new_line;
           end DISPLAY_STEP_RECORD;

          -- print all the records in the STEP list.
          procedure PRINT_ALL_STEP_RECORDS(HEAD :in LINK ) is
           CURRENT : LINK := HEAD ;
           begin
              while CURRENT /= null loop
                 DISPLAY_STEP_RECORD (CURRENT) ;
                 CURRENT := CURRENT.NEXT ;
              end loop;
           end PRINT_ALL_STEP_RECORDS ;
```

```
          -- print all the records in the READY QUEUE.
          procedure PRINT_ALL_QUEUE_RECORDS(HEAD:in LINK ) is
           CURRENT : LINK := HEAD ;
           begin
              while CURRENT /= null loop
                 DISPLAY_STEP_RECORD (CURRENT) ;
                 CURRENT := CURRENT.NEXT_READY ;
              end loop;
           end PRINT_ALL_QUEUE_RECORDS ;
--*************************************************************

       -- DISPLAY THE MAIN MENU.
       procedure SCHEDULER_MENU is
           begin
               new_line;
             set_col(25); put("MAIN MENU"); new_line;
             set_col(25); put("----------"); new_line(2);
             set_col(10); put("[1] schedule steps according
to their deadlines");
             new_line;
             set_col(10); put("[2] schedule steps according
to their start time");
             new_line;
             set_col(10); put("[3] schedule steps according
to (deadline + start time)");
             new_line;
             set_col(10); put("[4] schedule steps according
to their laxity ");
             new_line;
             set_col(10); put("[5] Print schedule");
             new_line;
             set_col(10); put("[6] Print ready queue");
             new_line;
            set_col(10); put("[7] Print a particular step");
             new_line;
             set_col(10); put("[8] Print step list");
             new_line;
             set_col(10); put("[9] Quit"); new_line(3);
           set_col(10); put("Enter the number of your choice
:  ");

           end SCHEDULER_MENU ;
--*************************************************************
          -- Linking a step to the ready list in order of its
          -- deadline.
          procedure INSERT_ORDER_DEADLINE(
```

```
                                R_QUEUE : in out LINK;
                                 A_RECORD  : in LINK ) is
            CURRENT  : LINK := R_QUEUE ;
            PREVIOUS : LINK := null ;
            LARGER_FOUND : BOOLEAN := FALSE ;
            begin
             while CURRENT /= null and not LARGER_FOUND loop
              if CURRENT.DEADLINE > A_RECORD.DEADLINE then
                 LARGER_FOUND := TRUE  ;
              else
                 PREVIOUS := CURRENT ;
                 CURRENT  := CURRENT.NEXT_READY ;
              end if ;
              end loop;
              A_RECORD.NEXT_READY := CURRENT ;
              if PREVIOUS = null then
                 R_QUEUE := A_RECORD ;
              else
                 PREVIOUS.NEXT_READY := A_RECORD ;
              end if ;
            end INSERT_ORDER_DEADLINE;


--************************************************************
          -- Linking a step to the ready list in order of its
          -- start time.
          procedure INSERT_ORDER_START_TIME(R_QUEUE,
                                A_RECORD : in out LINK ) is
            CURRENT  : LINK := R_QUEUE ;
            PREVIOUS : LINK := null ;
            LARGER_FOUND : BOOLEAN := FALSE ;
            begin
             while CURRENT /= null and not LARGER_FOUND loop
              if CURRENT.EARLIEST_START_TIME >
                                A_RECORD.EARLIEST_START_TIME
              then
                   LARGER_FOUND := TRUE  ;
              else
                   PREVIOUS := CURRENT ;
                   CURRENT  := CURRENT.NEXT_READY ;
              end if ;
              end loop;
              A_RECORD.NEXT_READY := CURRENT ;
              if PREVIOUS = null then
                 R_QUEUE := A_RECORD ;
              else
```

```
                PREVIOUS.NEXT_READY := A_RECORD ;
              end if ;
            end INSERT_ORDER_START_TIME;

--************************************************************
          -- Linking a step to the ready list in order of its
          -- start time.
        procedure INSERT_PENDING_ORDER_START_TIME(R_QUEUE,
                                A_RECORD : in out LINK ) is
            CURRENT  : LINK := R_QUEUE ;
            PREVIOUS : LINK := null ;
            LARGER_FOUND : BOOLEAN := FALSE ;
            begin
             while CURRENT /= null and not LARGER_FOUND loop
                if CURRENT.EARLIEST_START_TIME >
                          A_RECORD.EARLIEST_START_TIME then
                   LARGER_FOUND := TRUE  ;
                else
                   PREVIOUS := CURRENT ;
                   CURRENT  := CURRENT.NEXT_PENDING ;
                end if ;
             end loop;
             A_RECORD.NEXT_PENDING := CURRENT ;
             if PREVIOUS = null then
                R_QUEUE := A_RECORD ;
             else
                PREVIOUS.NEXT_PENDING := A_RECORD ;
             end if ;
            end INSERT_PENDING_ORDER_START_TIME;

--************************************************************
          -- Linking a step to the ready list in order of its
          -- (DEADLINE + start time).
          procedure INSERT_ORDER_MIXED(R_QUEUE, A_RECORD : in
                                        out LINK ) is
            CURRENT  : LINK := R_QUEUE ;
            PREVIOUS : LINK := null ;
            LARGER_FOUND : BOOLEAN := FALSE ;
            begin
             while CURRENT /= null and not LARGER_FOUND loop
              if (CURRENT.EARLIEST_START_TIME +
                  CURRENT.DEADLINE)>
                  (A_RECORD.EARLIEST_START_TIME +
                  A_RECORD.DEADLINE)
              then LARGER_FOUND := TRUE  ;
              else
```

```
                    PREVIOUS := CURRENT ;
                    CURRENT  := CURRENT.NEXT_READY ;
                end if ;
                end loop;
                A_RECORD.NEXT_READY := CURRENT ;
                if PREVIOUS = null then
                   R_QUEUE := A_RECORD ;
                else
                   PREVIOUS.NEXT_READY := A_RECORD ;
                end if ;
             end INSERT_ORDER_MIXED;

-- ********************************************************
          -- Linking a step to the ready list in order of its
          -- laxity (deadline - (est + duration)).
        procedure INSERT_ORDER_LAXITY(R_QUEUE, A_RECORD :
                                        in out LINK ) is
            CURRENT  : LINK := R_QUEUE ;
            PREVIOUS : LINK := null ;
            LARGER_FOUND : BOOLEAN := FALSE ;
            begin
             while CURRENT /= null and not LARGER_FOUND loop
                if (CURRENT.DEADLINE -
(CURRENT.EARLIEST_START_TIME + CURRENT.ESTIMATED_DURATION))
> (A_RECORD.DEADLINE -(A_RECORD.EARLIEST_START_TIME +
A_RECORD.ESTIMATED_DURATION))
                then LARGER_FOUND := TRUE  ;
                else
                     PREVIOUS := CURRENT ;
                     CURRENT  := CURRENT.NEXT_READY ;
                end if ;
             end loop;
             A_RECORD.NEXT_READY := CURRENT ;
             if PREVIOUS = null then
                R_QUEUE := A_RECORD ;
             else
                PREVIOUS.NEXT_READY := A_RECORD ;
             end if ;
          end INSERT_ORDER_LAXITY;

--  ********************************************************
          -- Linking a schedule record to the tail of the
          -- schedule list
          procedure INSERT_NEW_SCHEDULE_RECORD(LIST ,TAIL,
                                  A_RECORD : in out LINK1) is
             begin
```

```
              if LIST = null
             then A_RECORD.NEXT := LIST;
                  LIST := A_RECORD ;
              else
                    TAIL.NEXT := A_RECORD ;
              end if;
             TAIL := A_RECORD ;
          end INSERT_NEW_SCHEDULE_RECORD;
--  ********************************************************
          -- Linking a schedule record according to its
          -- expertise level
          procedure INSERT_ORDER_EXP_LEVEL (HEAD , A_RECORD :
                                        in out LINK1 ) is
            CURRENT  : LINK1 := HEAD ;
            PREVIOUS : LINK1 := null;
            LARGER_FOUND : BOOLEAN := FALSE ;
            begin
             while CURRENT /= null and not LARGER_FOUND loop
              if CURRENT.DESIGNER_LEVEL >
                             A_RECORD.DESIGNER_LEVEL then
                 LARGER_FOUND := TRUE  ;
              else
                 PREVIOUS := CURRENT ;
                 CURRENT  := CURRENT.NEXT ;
              end if ;
              end loop;
              A_RECORD.NEXT := CURRENT ;
              if PREVIOUS = null then
                 HEAD     := A_RECORD ;
              else
                 PREVIOUS.NEXT := A_RECORD ;
              end if ;
            end INSERT_ORDER_EXP_LEVEL;
--************************************************************
          -- Linking a schedule_record to the schedule in
          -- order of its start time.
          procedure INSERT_ORDER_START_TIME (HEAD , A_RECORD
                                        : in out LINK1 ) is
            CURRENT  : LINK1 := HEAD ;
            PREVIOUS : LINK1 := null;
            LARGER_FOUND : BOOLEAN := FALSE ;
            begin
             while CURRENT /= null and  not LARGER_FOUND loop
           if CURRENT.START_TIME >= A_RECORD.START_TIME  then
                 LARGER_FOUND := TRUE  ;
              else
```

```
                PREVIOUS := CURRENT ;
                CURRENT  := CURRENT.NEXT ;
             end if ;
             end loop;
             A_RECORD.NEXT := CURRENT ;
             if PREVIOUS = null then
                HEAD := A_RECORD ;
             else
                PREVIOUS.NEXT := A_RECORD ;
             end if ;
          end INSERT_ORDER_START_TIME;
--*************************************************************
          -- Linking a schedule_record to the schedule in
          -- order of its step id.
          procedure INSERT_ORDER_STEP_ID (HEAD , A_RECORD
                                          : in out LINK1 ) is
            CURRENT  : LINK1 := HEAD ;
            PREVIOUS : LINK1 := null;
            LARGER_FOUND : BOOLEAN := FALSE ;
            begin
             while CURRENT /= null and not LARGER_FOUND loop
              if CURRENT.STEP_ID >= A_RECORD.STEP_ID then
                   LARGER_FOUND := TRUE  ;
              else
                   PREVIOUS := CURRENT ;
                   CURRENT  := CURRENT.NEXT ;
             end if ;
             end loop;
             A_RECORD.NEXT := CURRENT ;
             if PREVIOUS = null then
                HEAD := A_RECORD ;
             else
                PREVIOUS.NEXT := A_RECORD ;
             end if ;
          end INSERT_ORDER_STEP_ID;

--*************************************************************
      procedure LEVEL_MINMUM (MATRIX     : in DESIGNER_MATRIX;
                              LEVEL      : in EXPERTISE_LEVEL;
                                MAX_LEVEL  : in natural;
                                ROW_LENGTH : in vector ;
                               L, J       : in out natural) is
          MIN : natural;
          temp : natural := EXPERTISE_LEVEL'POS(LEVEL) + 1;
          begin
                    L := temp;
```

```
                J := 1;
              MIN := 1000;    -- MATRIX ( temp, 1);
              for k IN temp..MAX_LEVEL loop
                if ROW_LENGTH (k) /= 0 then
                for i IN 1..ROW_LENGTH (k) loop
                  if MIN > MATRIX( k,i) then
                     MIN := MATRIX ( k, i) ;
                         L := k;
                     J := i;
                  end if;
                end loop;
                end if;
              end loop;
        end LEVEL_MINMUM;
--************************************************************
      function ROW_MINMUM (MATRIX     : in DESIGNER_MATRIX;
                           LEVEL      : in EXPERTISE_LEVEL;
                           ROW_LENGTH : in vector) return
                                        natural is
        temp : natural := EXPERTISE_LEVEL'POS(LEVEL) + 1;
        MIN, J  : natural;
        begin
                  J:= 1;
                  MIN := MATRIX(temp, J);
                  for i in 1..ROW_LENGTH(temp) loop
                  if MATRIX( temp,i) > MIN  then
                     MIN := MATRIX ( temp, i) ;
                         J := i;
                  end if;
                  end loop;
                  if ROW_LENGTH(temp)=0 then
                   return 1000;
                  else
                       return J;
                  end if;
        end ROW_MINMUM;
--************************************************************
        -- Search for target step. Return the position and
        -- previous if found,
        -- " Record not found " if not found.
        procedure FIND_STEP (HEAD     : in LINK;
                             S_ID     : in  natural;
                             POSITION   : in out LINK;
                             PREVIOUS   : in out LINK;
                             FOUND      : in out boolean) is
```

```
            begin
               POSITION  := HEAD ;
               PREVIOUS  := null ;
               FOUND     := FALSE;
               while POSITION /= null and not FOUND loop
               if POSITION.STEP_ID = S_ID then
                  FOUND := TRUE ;
               else
                  PREVIOUS := POSITION ;
                  POSITION := POSITION.NEXT ;
               end if ;
               end loop ;
               if FOUND = FALSE then
                  put("STEP NOT FOUND ");
               end if ;
            end FIND_STEP ;
--*************************************************************
 -- Delete the step found by search and relinks the step list.
     procedure DELETE_FROM_STEP_LIST (HEAD     : in out LINK;
                                      POSITION : in LINK;
                                     PREVIOUS : in LINK) is
              begin
                 if HEAD = POSITION   then
                         HEAD := HEAD.NEXT;
                     else
                       PREVIOUS.NEXT := POSITION.NEXT;
                  end if;

            end DELETE_FROM_STEP_LIST;

 --*************************************************************
 -- Delete the step found by search and relinks the step list.
     procedure DELETE_FROM_PENDING_LIST (HEAD  : in out LINK;
                                      POSITION : in LINK;
                                    PREVIOUS : in LINK) is

            begin
                 if HEAD = POSITION   then
                     HEAD := HEAD.NEXT_PENDING;
                  else
                    PREVIOUS.NEXT_PENDING :=
                     POSITION.NEXT_PENDING;
                  end if;

            end DELETE_FROM_PENDING_LIST;
```

```
--*************************************************************
       -- Delete a step from the schedule.
      procedure DELETE_FROM_SCHEDULE (HEAD     : in out LINK1;
                                      POSITION : in LINK1;
                                     PREVIOUS : in LINK1) is

           begin
                if HEAD = POSITION   then
                    HEAD := HEAD.NEXT;
                 else
                   PREVIOUS.NEXT := POSITION.NEXT;
                 end if;

           end DELETE_FROM_SCHEDULE;
--*************************************************************
          -- Delete a step from the head of the ready list.
          procedure DELETE_FROM_READY_QUEUE(HEAD:
                                            in out LINK) is

           begin
               HEAD := HEAD.NEXT_READY;

           end DELETE_FROM_READY_QUEUE;
--*************************************************************
          -- Decrementing the in_degree of the successors of
          -- a step and adjusting its earliest start time.
       procedure DECREMENT_IN_DEGREE (STEP, LIST :in out LINK;
                                      finish_t :in natural) is
            POSITION : LINK := LIST;
            t : set := STEP.SUCCESSORS;
            k : natural ;
              k1 : natural := 0;
            FOUND : boolean := FALSE;
            begin
              if size(t) /= 0 then
              while POSITION /= null  and k1 <= size(t) loop
               k := POSITION.STEP_ID;
               if member(k, t) then
                if POSITION.EARLIEST_START_TIME < finish_t
               then POSITION.EARLIEST_START_TIME := finish_t;
                end if;
               POSITION.IN_DEGREE := POSITION.IN_DEGREE - 1;
                    k1 := k1 + 1;
               end if;
               POSITION := POSITION.NEXT;
```

```
            end loop;
              end if;
            end DECREMENT_IN_DEGREE;
--*************************************************************
        function list_size(LIST : in LINK)return natural is
              current : link := LIST;
              K       : natural := 0;
              begin
                while current /= null loop
                      K := K + 1;
                      current := current.NEXT;
                end loop;
                return K;
              end list_size;

--*************************************************************
          -- checking the in_degree of the successors of the
          -- Tassigned step this works with deadline
          -- heuristic.

          procedure CHECK_IN_DEGREE(
                    STEP, R_QUEUE , LIST  : in out LINK;
                    finish_t        : in natural) is
             POSITION : LINK := LIST;
             PREVIOUS : LINK := null ;
             t : set := STEP.SUCCESSORS;
             t1:  set;
             k : natural;
             FOUND : boolean := FALSE;
             begin
               if size(t) /= 0 then
               while POSITION /= null  loop
                    k := POSITION.STEP_ID;
                    if member(k, t) then
                    if POSITION.EARLIEST_START_TIME <
                                        finish_t
                    then POSITION.EARLIEST_START_TIME :=
                                        finish_t;
                    end if;
              POSITION.IN_DEGREE := POSITION.IN_DEGREE - 1;
              if POSITION.IN_DEGREE = 0
              then
                INSERT_PENDING_ORDER_START_TIME (R_QUEUE,
                                        POSITION);
                    POSITION := POSITION.NEXT;
              else
```

```
                    PREVIOUS := POSITION;
                        POSITION := POSITION.NEXT;
                                end if;
                             else
                                PREVIOUS := POSITION;
                           POSITION := POSITION.NEXT;
                             end if;
                       end loop;
                    end if;
            end CHECK_IN_DEGREE;

--*************************************************************
        -- checking the in_degree of the successors of the
        -- assigned step
        -- This works with start time heuristic.

      procedure CHECK_IN_DEGREE_1 (STEP, LIST  : in out LINK;
                                   R_QUEUE : in out LIST_VECTOR;
                                   finish_t       : in natural) is
          POSITION : LINK := LIST;
          PREVIOUS : LINK := null ;
          t : set := STEP.SUCCESSORS;
          t1:  set;
          k, k1 : natural;
          FOUND : boolean := FALSE;
          begin
              if size(t) /= 0 then
                while POSITION /= null  loop
                      k := POSITION.STEP_ID;
                      if member(k, t) then
                        if POSITION.EARLIEST_START_TIME <
                                      finish_t
                        then POSITION.EARLIEST_START_TIME
                                      := finish_t;
                         end if;
              POSITION.IN_DEGREE := POSITION.IN_DEGREE - 1;
               if POSITION.IN_DEGREE = 0
               then
              k1 :=
              EXPERTISE_LEVEL'POS(POSITION.EXP_LEVEL) + 1;
           INSERT_ORDER_START_TIME (R_QUEUE(k1), POSITION);
                                POSITION := POSITION.NEXT;
                           else
                                PREVIOUS := POSITION;
                                POSITION := POSITION.NEXT;
                           end if;
```

```
                    else
                       PREVIOUS := POSITION;
                       POSITION := POSITION.NEXT;
                    end if;
                 end loop;
              end if;
           end CHECK_IN_DEGREE_1;

--*************************************************************
       -- checking the in_degree of the successors of the
      -- assigned step. This works with start time heuristic.

      procedure CHECK_IN_DEGREE_2 (STEP, LIST  : in out LINK;
                          R_QUEUE      : in out LIST_VECTOR;
                            finish_t        : in natural) is
           POSITION : LINK := LIST;
           PREVIOUS : LINK := null ;
           t : set := STEP.SUCCESSORS;
           t1:  set;
           k, k1 : natural;
           FOUND : boolean := FALSE;
           begin
               if size(t) /= 0 then
                 while POSITION /= null  loop
                       k := POSITION.STEP_ID;
                       if member(k, t) then
                         if POSITION.EARLIEST_START_TIME <
                                        finish_t
                         then POSITION.EARLIEST_START_TIME
                                        := finish_t;
                          end if;
                      POSITION.IN_DEGREE := POSITION.IN_DEGREE - 1;
                       if POSITION.IN_DEGREE = 0
                       then
                    k1 := EXPERTISE_LEVEL'POS(POSITION.EXP_LEVEL)
                       + 1;
             INSERT_ORDER_START_TIME (R_QUEUE(k1), POSITION);
                                POSITION := POSITION.NEXT;
                          else
                                PREVIOUS := POSITION;
                                POSITION := POSITION.NEXT;
                          end if;
                       else
                          PREVIOUS := POSITION;
                          POSITION := POSITION.NEXT;
                       end if;
```

```
                end loop;
              end if;
          end CHECK_IN_DEGREE_2;
--*************************************************************
          -- checking the pending list for ready steps of a
          -- certain level and insert them into the
          -- corresponding ready_list according to their
          -- deadlines

        procedure GET_READY_STEPS (
                    t ,k : in natural;
                     LIST : in out LINK;
                     R_QUEUE : in out LIST_VECTOR) is

            POSITION : LINK := LIST;
            PREVIOUS : LINK := null ;
            temp : natural;
            FOUND : boolean := FALSE;
            begin
              while POSITION /= null  loop
             temp := EXPERTISE_LEVEL'POS(POSITION.EXP_LEVEL)
                                          + 1;
              if POSITION.EARLIEST_START_TIME <=  t and temp
                                          >= k
               then
              INSERT_ORDER_DEADLINE (R_QUEUE(temp),POSITION);
               DELETE_FROM_PENDING_LIST(LIST,POSITION,
                                          PREVIOUS);
                POSITION := POSITION.NEXT_PENDING;
                else
                PREVIOUS := POSITION;
                POSITION := POSITION.NEXT_PENDING;
                end if;
                end loop;
            end GET_READY_STEPS;
--*************************************************************
          -- checking the pending list for ready steps of a
          -- certain level and insert them into the
          -- corresponding ready_list according to
          -- their (deadlines + start times).
        procedure GET_READY_STEPS_1 (t ,k : in natural;
                          LIST,  R_QUEUE    : in out LINK) is

            POSITION : LINK := LIST;
            PREVIOUS : LINK := null ;
            temp : natural;
```

```
            FOUND : boolean := FALSE;
            begin
             while POSITION /= null  loop
            temp:= EXPERTISE_LEVEL'POS(POSITION.EXP_LEVEL)+1;
             if POSITION.EARLIEST_START_TIME <=  t and temp = k
              then
               INSERT_ORDER_MIXED (R_QUEUE, POSITION);
               DELETE_FROM_PENDING_LIST(LIST,POSITION,
                                        PREVIOUS);
               POSITION := POSITION.NEXT_PENDING;
              else
               PREVIOUS := POSITION;
               POSITION := POSITION.NEXT_PENDING;
            end if;
            end loop;
            end GET_READY_STEPS_1;
--************************************************************
          -- checking the pending list for ready steps of a
          -- certainlevel and insert them into the
          -- corresponding ready_list  according to their
          -- LAXITY.
        procedure GET_READY_STEPS_2 (t ,k : in natural;
                              LIST,  R_QUEUE : in out LINK) is

            POSITION : LINK := LIST;
            PREVIOUS : LINK := null ;
            temp : natural;
            FOUND : boolean := FALSE;
            begin
              while POSITION /= null  loop
              temp:=
               EXPERTISE_LEVEL'POS(POSITION.EXP_LEVEL) + 1;
            if POSITION.EARLIEST_START_TIME <=  t and temp = k
             then
              INSERT_ORDER_LAXITY (R_QUEUE, POSITION);
              DELETE_FROM_PENDING_LIST(LIST,POSITION,
                                                  PREVIOUS);
              POSITION := POSITION.NEXT_PENDING;
             else
              PREVIOUS := POSITION;
              POSITION := POSITION.NEXT_PENDING;
             end if;
            end loop;
            end GET_READY_STEPS_2;

--************************************************************
```

```
        -- get the top steps in the ready list and
        -- insert them into the corresponding ready_list
        -- according to their deadlines

        procedure GET_READY_STEPS (LIST   :in out LINK;
                                   R_QUEUE :in out
                                       LIST_VECTOR) is

            POSITION : LINK := LIST;
            PREVIOUS : LINK := null;
            temp : natural;
            temp1 : natural;
            begin
                   if POSITION /= null then
                     temp := POSITION.EARLIEST_START_TIME;
                   end if;
                    while POSITION /= null and then
                  POSITION.EARLIEST_START_TIME = temp  loop
                  temp1 :=
                  EXPERTISE_LEVEL'POS(POSITION.EXP_LEVEL)
                            +1;
                 INSERT_ORDER_START_TIME (R_QUEUE(temp1),
                                                POSITION);

                  DELETE_FROM_PENDING_LIST(LIST,POSITION,
                                                PREVIOUS);
                  POSITION := POSITION.NEXT_PENDING;
                    end loop;
            end GET_READY_STEPS;
--************************************************************
        -- get the top steps in the ready list and
        -- insert them into the corresponding ready_list
        -- according to their (deadlines + start times).

        procedure GET_READY_STEPS_1 (LIST :in out LINK;
                              R_QUEUE : in out LIST_VECTOR) is

            POSITION : LINK := LIST;
            PREVIOUS : LINK := null;
            temp : natural;
            temp1 : natural;
            begin
                 if POSITION /= null then
                    temp := POSITION.EARLIEST_START_TIME ;
                 end if;
```

```
                while POSITION /= null and then
                  POSITION.EARLIEST_START_TIME = temp  loop
                  temp1 :=
                  EXPERTISE_LEVEL'POS(POSITION.EXP_LEVEL)
                                        + 1;
                   INSERT_ORDER_MIXED (R_QUEUE(temp1),
                                        POSITION);
                  DELETE_FROM_PENDING_LIST(LIST,POSITION,
                                                 PREVIOUS);
                   POSITION := POSITION.NEXT_PENDING;
                end loop;
          end GET_READY_STEPS_1;
--*************************************************************
        -- get the top steps in the ready list and
        -- insert them into the corresponding ready_list
        -- according to their Laxity.

        procedure GET_READY_STEPS_2 (LIST      :in out LINK;
                                R_QUEUE: in out LIST_VECTOR) is

            POSITION : LINK := LIST;
            PREVIOUS : LINK := null;
            temp : natural;
            temp1 : natural;
            begin
                 if POSITION /= null then
                    temp := POSITION.EARLIEST_START_TIME ;
                 end if;
                  while POSITION /= null and then
                    POSITION.EARLIEST_START_TIME = temp  loop
                    temp1 :=
                    EXPERTISE_LEVEL'POS(POSITION.EXP_LEVEL)
                                        + 1;
                       INSERT_ORDER_LAXITY (R_QUEUE(temp1),
                                        POSITION);
                    DELETE_FROM_PENDING_LIST(LIST,POSITION,
                                        PREVIOUS);
                    POSITION := POSITION.NEXT_PENDING;
                  end loop;
          end GET_READY_STEPS_2;

--*************************************************************
        procedure SUGGEST_DEADLINE_SLIP(STEP  : in out link;
                                      VALUE :in natural) is
            answer :character := 'n';
```

```
        begin
            put("in-feasible schedule: step # ");
            test_io_pkg.put (STEP.STEP_ID); new_line;
            put("suggested deadline should be >= ");
            test_io_pkg.put (VALUE); new_line;
        put ("Would you like to change it?  Answer(y/n)");
            get(answer);new_line;
            if answer = 'y' or answer = 'Y'
            then
                put ("Enter the new Deadline ");
                get (STEP.deadline);
                STEP.DEADLINE_CHANGE := TRUE;
            else
                put_line("INFEASIBLE SCHEDULE");
            end if;
         end SUGGEST_DEADLINE_SLIP;
--*********************************************************

        -- checking the feasibility of the schedule with
        -- each step in the ready queue
        procedure STRONGLY_FEASIBLE (R_QUEUE     : in LINK;
                                     MATRIX      : in
                                           DESIGNER_MATRIX;
                                     D_VECTOR    : in vector;
                                     MAX_LEVEL   : in natural;
                                   FEASIBLE    : in out boolean ) is
          CURRENT  : LINK := R_QUEUE ;
          PREVIOUS : LINK := null;
          temp, temp1       : natural;
            J       : natural := 1;
            L       : natural := 1;
          MIN : natural :=0 ;
          begin
            FEASIBLE := TRUE ;
              while CURRENT /= null and FEASIBLE loop
                   LEVEL_MINMUM
                  (MATRIX,CURRENT.EXP_LEVEL,MAX_LEVEL,
                                   D_VECTOR,L,J );
        --           put_line (" pass 10  ");
                   MIN := MATRIX(L,J);
                   if MIN >=  CURRENT.EARLIEST_START_TIME
                   then   temp := MIN;
                   else     temp :=
                            CURRENT.EARLIEST_START_TIME;
                   end if;
```

```
                    temp := temp +
                            CURRENT.ESTIMATED_DURATION;
                    IF temp > CURRENT.DEADLINE
                    then
                       SUGGEST_DEADLINE_SLIP(CURRENT,temp);
                 -- FEASIBLE := FALSE;
                    end if;
                    if temp > CURRENT.DEADLINE then
                            FEASIBLE := FALSE;
                  end if;
                    PREVIOUS := CURRENT ;
                CURRENT  := CURRENT.NEXT_READY ;
            end loop;
            end STRONGLY_FEASIBLE;

--************************************************************
      -- Assign a step to a designer according to its deadline
       -- and its expertise level
       procedure ASSIGN_STEP  (R_QUEUE : in out LIST_VECTOR;
                               LIST     : in out  LINK;
                               MATRIX   : in out
                                          DESIGNER_MATRIX;
                               ROW_LENGTH    : in vector;
                               M,N          : in out natural;
                               L             : in natural;
                                SCH           : in out LINK1;
                               done         : out boolean ) is
          K,  finish     : natural;
          MIN : natural := MATRIX(M,N);
          MIN1: natural;
          J   : natural := 1;
          temp : natural := 0;
          temp1 : link := R_QUEUE(L);
          begin
                    done := TRUE;
                    K :=
                   EXPERTISE_LEVEL'POS(temp1.EXP_LEVEL)+ 1;
                   J := ROW_MINMUM(MATRIX, temp1.EXP_LEVEL,
                               ROW_LENGTH);
               if j /= 1000 then
                        MIN1 := MATRIX(K,J);
               else MIN1 := 1000;
               end if;
                    if MIN1 <= temp1.EARLIEST_START_TIME
                   then
                       temp := temp1.EARLIEST_START_TIME;
```

```
                    finish := temp +
                              temp1.ESTIMATED_DURATION;
                    MATRIX (K,J):= finish;
                    M := K;
                    N := J;
                    CREATE_SCHEDULE_RECORD(temp1.STEP_ID,temp,
                    finish,J,temp1.EXP_LEVEL,
                    EXPERTISE_LEVEL'VAL(K-1));
                    INSERT_ORDER_START_TIME(SCH,
                              NEW_S_RECORD);
                    DELETE_FROM_READY_QUEUE(R_QUEUE(L));
          elsif ((M > K) and then (R_QUEUE(M)/= null) and then
                (R_QUEUE(M).EARLIEST_START_TIME <= MIN)) then
                  done := FALSE;
          else
              if MIN >=  temp1.EARLIEST_START_TIME
              then   temp := MIN;
              elsetemp := temp1.EARLIEST_START_TIME;
              end if;
          finish := temp + temp1.ESTIMATED_DURATION;
          MATRIX (M,N):= finish;
          CREATE_SCHEDULE_RECORD (temp1.STEP_ID,
          temp, finish,N,temp1.EXP_LEVEL,
          EXPERTISE_LEVEL'VAL(M-1));
          INSERT_ORDER_START_TIME(SCH, NEW_S_RECORD);
          DELETE_FROM_READY_QUEUE(R_QUEUE(L));
          end if;
      end ASSIGN_STEP;
--************************************************************
          -- reset the step list and the schedule for
          -- incrementing the schedule
          -- with new steps at certain time
          procedure RESET_FOR_RESCHEDULING(
                              d_list : in out D_LINK;
                              LIST     : in out LINK;
                              MATRIX       : in out
                                        designer_matrix;
                              d_vector     : in vector) is
           M,N,index, length          : natural;
           step_id, finish_t :natural;
           designer      : string(1..16);
           data_file2   : FILE_TYPE;
           CURRENT      :LINK;
           begin
           OPEN(data_file2, IN_FILE, "temp3");
              WHILE not END_OF_FILE(data_file2) loop
```

```
                get(data_file2,step_id);skip_line(data_file2);
--                test_io_pkg.put(step_id);
                  get(data_file2,finish_t);
--                test_io_pkg.put(finish_t);
                  skip_line(data_file2);
                  get_line(data_file2,designer,length);
                       for i in length+1..16 loop
                           designer(i) :=' ';
                       end loop;
--                  put(designer);
--                  new_line;
--                  skip_line(data_file2);
                  CURRENT := LIST;
                  while CURRENT /= null loop
                       if member(step_id,CURRENT.PREDECESSORS)
                       then
                      CURRENT.IN_DEGREE:= CURRENT.IN_DEGREE-1;
                      if CURRENT.EARLIEST_START_TIME <finish_t
                        then CURRENT.EARLIEST_START_TIME :=
                                              finish_t;
                        end if;
                    end if;
                      find_designer_position(d_list,index,
                                designer);
                        if index <=d_vector(1)
                        then M := 1;
                             N := index;
                        elsif index > d_vector(1) and index
                                <=(d_vector(1)+d_vector(2))
                        then M := 2;
                             N := index - d_vector(1);
                       elsif index > (d_vector(1)+ d_vector(2))
                        then M := 3;
                       N := index - (d_vector(1)+ d_vector(2));
                        end if;
                            if MATRIX (M,N)<finish_t
                            then
                           MATRIX (M,N):=finish_t;
                            end if;
                            CURRENT  := CURRENT.NEXT;
                    end loop;
                  end loop;
                     CLOSE(data_file2);
--                  DELETE(data_file2);
                  exception
                     when NAME_ERROR=>
```

```
                    put_line("No original schedule");
                end RESET_FOR_RESCHEDULING;

            -- Search for target step. Return the position and
            -- previous if found,
          procedure FIND_DESIGNER_POSITION(HEAD    : in D_LINK;
                                           D_INDEX : out natural;
                                           d_name  : in string) is
                POSITION  : D_LINK := HEAD ;
                K : natural :=1;
            begin
             while POSITION /= null and then POSITION.D_name /=
                                             d_name loop
                    POSITION := POSITION.NEXT ;
                    k := k+1;
                end loop ;
             D_INDEX := k ;
             end FIND_DESIGNER_POSITION ;

      end scheduler;
```

3. Main Programs

new_ecs.a (Manager_Interface) ************************************

```
-- Title              : Manager_Interface main program
-- Author             : Salah badr
-- Date               : 25 August 1993
-- Revised            :
-- System             : Suns7
-- Compiler           : VerdixAda
-- Description        :

          with ECS_OPERATIONS;  use ECS_OPERATIONS;
          with TEXT_IO; -- BASIC_NUM_IO;
          use TEXT_IO;  -- BASIC_NUM_IO;
          -- Main program.
          package ecs_manager is

Name1           : STRING(1..9) := "supportDB";
```

```
option          : STRING(1..1) ;
Name3           : STRING(1..64);
Name4           : STRING(1..64);
Name5           : STRING(1..64);
Name6           : STRING(1..64);
Name7           : STRING(1..64);
Name8           : STRING(1..64);
Name9           : STRING(1..64);
NameA           : STRING(1..64);
NameB           : STRING(1..64);
NameC           : STRING(1..64);
NameD           : STRING(1..64);
NameE           : STRING(1..64);
a_name          : STRING(1..64);
an_id           : STRING(1..5);
SELECTOR        : natural := 0;
Length          : integer;
answer          : character:='y';
data_file       : FILE_TYPE;    -- logical file definitions
data_file1      : FILE_TYPE;    -- logical file definitions

--************************************************************

procedure ECS1;

end ecs_manager;

package body ecs_manager is


procedure ECS1 is

          package nat_io is new integer_io(natural);  use
                                nat_io;

begin

          put_line("ECS has been entered.");
          put("Name3 =>  ");
          put_line(Name3);
          put("SELECTOR =>  ");
          put(SELECTOR);
          new_line;

          case SELECTOR  is
             -- Show the prototypes in the database.
```

```
when 1 =>   --show prototypes
     option:="8";
    Show_Prototypes(Name1, option);
 when 2 =>   -- show steps
     option:="3";
     Show_Step(Name1,option,Name3);
when 3 =>    -- show step
       option:="2";
       Show_Step(Name1,option,Name3);
when 4 =>     -- show schedule
      option:="2";
      Show_Schedule(Name1,option);
when 5 =>      --create prototype
     option:="1";
     put("Enter prototype's name: ");
     get_line(Name3,Length);
     for i in Length+1..64 loop
          Name3(i):=' ';
      end loop;
      Create_Prototype(Name1,option,Name3);
       put("More subcomponents to add[answer Y/
                 N]: ");
        get(answer);
        skip_line;
        while answer='y' or answer='Y' loop
           put("Enter component name: ");
           get_line(Name5,Length);
           for i in Length+1..64 loop
               Name5(i):=' ';
           end loop;
          put("Enter parent component name: ");
           get_line(Name7,Length);
           for i in Length+1..64 loop
               Name7(i):=' ';
           end loop;
           Add_SubComponent(Name1,"4",Name3,
                       "1:1",Name5,"1:1",Name7);
         put("More subcomponents to add[answer
                Y/N]: ");
           get(answer);
           skip_line;
         end loop;
 when 6 =>    --create step
      option:="1";
    Create_Step(Name1,option,Name3,Name4);
 when 7 =>     -- edit step
```

```
    system_call("mainstep "&Name1&" 4 "&Name3&"
       "&NameE&" "&Name4&" "&Name5&" "&Name6&"
       "&Name7&" "&Name8&" "&Name9&" "&NameA&"
       "&NameB&" "&NameC&" "&NameD&" > temp5");

      OPEN(data_file, IN_FILE,"temp5");
      get_line(data_file,a_name,length);
        for i in Length+1..5 loop
            a_name(i):=' ';
        end loop;
      if a_name(1) ='s' then
         put_designers(Name1,"2");
         get_sched_data(Name1,"0");
         get_sched_data_1(Name1,"0");
         main(1);
          elsif a_name(1) ='c' then
     put_line("Cannot Update a Completed Step");
      elsif a_name(1) ='a' then
    put_line("Cannot Update an Abandoned Step");
      end if;
      CLOSE(data_file);
when 8 =>      -- approve step
      option:="c";
      Show_Step(Name1,option,Name3);
when 9 =>      -- schedule step
      put_designers(Name1,"2");
          get_sched_data(Name1,Name3);
          get_sched_data_1(Name1,"0");
      main(1);
when 10 =>      -- commit step
      option:="e";
      Show_Step(Name1,option,Name3);
when 11 =>      -- suspend step
     begin
system_call("mainstep "&Name1&" i "&Name3&" 1 >
                                        temp");
      OPEN(data_file1, IN_FILE,"temp");
      WHILE not END_OF_FILE(data_file1) loop
        get_line(data_file1,an_id,length);
        for i in Length+1..5 loop
            an_id(i):=' ';
        end loop;
        get_line(data_file1,Name4,length);
        for i in Length+1..64 loop
            Name4(i):=' ';
         end loop;
```

```
        auto_mail2(Name4,an_id);
      end loop;
      put_designers(Name1,"2");
      get_sched_data(Name1,"0");
      get_sched_data_1(Name1,"0");
      main(0);
      CLOSE(data_file1);
   exception
      when NAME_ERROR=>
      put_line("No ready assignment for this
                designer");
    end;
when 12 =>      -- abandon step
    begin
system_call("mainstep "&Name1&" i "&Name3&" 5 >
                                        temp");
      OPEN(data_file1, IN_FILE,"temp");
      WHILE not END_OF_FILE(data_file1) loop
        get_line(data_file1,an_id,length);
        for i in Length+1..5 loop
            an_id(i):=' ';
        end loop;
        get_line(data_file1,Name4,length);
        for i in Length+1..64 loop
            Name4(i):=' ';
         end loop;
        auto_mail3(Name4,an_id);
      end loop;
      put_designers(Name1,"2");
      get_sched_data(Name1,"0");
      get_sched_data_1(Name1,"0");
      main(0);
      CLOSE(data_file1);
   exception
      when NAME_ERROR=>
      put_line("No ready assignment for this
                designer");
    end;
when 13 =>    -- remove transfer file
      system_call("rm ddbdisplay");
when 14 => -- add designer
      option:="1";
      for i in 25..64 loop
          Name3(i):=' ';
      end loop;
      for i in 2..64 loop
```

```
            Name4(i):=' ';
        end loop;
        Add_designer(Name1,option,Name3,Name4);
      put_designers(Name1,"2");
      get_sched_data(Name1,"0");
      get_sched_data_1(Name1,"0");
      main(0);
 when 15 =>     -- show designers
      option:="2";
      Show_designer(Name1,option);
 when 16 =>     -- Delete designer
      option:="3";
      for i in 25..64 loop
          Name3(i):=' ';
      end loop;

      -- if status= busy, reschedule
      Delete_designer(Name1,option,Name3);
      --and reschdule
      if Name4(1..4) = "Busy" then
      put_line("NOTICE:  The Designer just
                deleted was busy");
     put_line(" RESCHEDULING his/her tasks.");
      get_sched_data_2(Name1,Name3);
   system_call("mainsched "&Name1&" 7 "&Name3);
      put_designers(Name1,"2");
      get_sched_data_1(Name1,Name3);
      main(0);
      end if;

 when 17 =>        -- Change expertise level
      option:="4";
      for i in 25..64 loop
          Name3(i):=' ';
      end loop;
      for i in 2..64 loop
          Name4(i):=' ';
      end loop;
    Change_exp_level(Name1,option,Name3,Name4);
      put_designers(Name1,"2");
      get_sched_data(Name1,"0");
      get_sched_data_1(Name1,"0");
      main(1);

-- exception handling for selector case.
when others =>
```

```
                put(" BAD CHOICE. PLEASE TRY AGAIN");
                new_line ;
            end case;
end  ECS1;

begin

  null;

end ecs_manager;
```

Designer_Interface.a **********

```
-- Title              : Designer_Interface program
-- Author             : Salah badr
-- Date               : 25 September 1993
-- Revised            :
-- System             : Suns7
-- Compiler           : VerdixAda
-- Description        :

    with ECS_OPERATIONS;  use ECS_OPERATIONS;
      with TEXT_IO; -- BASIC_NUM_IO;
      use TEXT_IO; --, BASIC_NUM_IO ;
    -- Main program.
    procedure DESIGNER_INTERFACE is

      package nat_io is new integer_io(natural);  use nat_io;

Name1           : STRING(1..9) := "supportDB";
option          : STRING(1..1) ;
Name3           : STRING(1..64);
Name4           : STRING(1..64);
Name6           : STRING(1..64);
Name5           : STRING(1..64);
Name7           : STRING(1..64);
an_id           : STRING(1..5);
the_time        :string(1..14);
SELECTOR        : natural := 0;
Length          : integer;
data_file       : FILE_TYPE;     -- logical file definitions
data_file1      : FILE_TYPE;     -- logical file definitions
answer          :character :='y';
--*************************************************************

       -- DISPLAY THE MAIN MENU.
```

```
        procedure DESIGNER_MENU is
            begin
              new_line;
              set_col(25); put("DESIGNER MENU"); new_line;
              set_col(25); put("=============="); new_line(2);
              set_col(5); put("[1]  Show Prototypes");
              new_line;
              set_col(5); put("[2]  Show Component Subtree");
              new_line;
              set_col(5); put("[3]  Show Steps");
              new_line;
              set_col(5); put("[4]  Show Step");
              new_line;
              set_col(5); put("[5]  Create Prototype");
              new_line;
--              set_col(5); put("[6] Create Step");
--              new_line;
              set_col(5); put("[6]  Create Substep");
              new_line;
              set_col(5); put("[7]  Commit Substep");
              new_line;
              set_col(5); put("[8]  Retrieve Version");
              new_line;
              set_col(5); put("[9]  Retrieve Spec File");
              new_line;
              set_col(5); put("[10] Retrieve Imp File");
              new_line;
                set_col(5); put("[11] Show Schedule");
              new_line;
              set_col(5); put("[12] Quit"); new_line(3);
             set_col(5); put("Enter the number of your choice
:  ");

            end DESIGNER_MENU ;
--*************************************************************

begin
   loop
     begin
        system_call("mainstep "&Name1&" h >temp");
        OPEN(data_file, IN_FILE,"temp");
        get_line(data_file,an_id,length);
        for i in Length+1..5 loop
          an_id(i):=' ';
        end loop;
        if an_id(1) ='F' then
```

```
            get_line(data_file,Name4,length);
            for i in Length+1..64 loop
                Name4(i):=' ';
            end loop;
              system_call("mkdir "&Name4);
            get_line(data_file,an_id,length);
            for i in Length+1..5 loop
                an_id(i):=' ';
            end loop;
            system_call("mainstep "&Name1&" g "&an_id);
        end if;
        CLOSE(data_file);
        system_call("rm temp");
        end;
--          loop
          DESIGNER_MENU ;
             get(SELECTOR); skip_line ;
           case SELECTOR  is
              -- Show the prototypes in the database.
              when 1 =>         -- Show Prototypes
                   option:="8";
                   Show_Prototypes(Name1,option);
                   system_call("more ddbdisplay");
              when 2 =>         -- Show Component Subtree
                   option:="3";
                    put("Enter prototype's name: ");
                    get_line(Name3,Length);
                    for i in Length+1..64 loop
                         Name3(i):=' ';
                    end loop;
                put("Enter variation and version[var:ver]: ");
                    get_line(Name6,Length);
                    for i in Length+1..64 loop
                        Name6(i):=' ';
                    end loop;
                    put("Enter component name: ");
                    get_line(Name5,Length);
                    for i in Length+1..64 loop
                        Name5(i):=' ';
                    end loop;

general_function(Name1,option,Name3,Name6,Name5);
              when 3 =>        -- Show Steps
                   option:="3";
                 put("Enter Step status or all to see all the
                                steps: ");
```

```
            get_line(Name3,Length);
             for i in Length+1..64 loop
                Name3(i):=' ';
            end loop;
            Show_Step(Name1,option,Name3);
            system_call("more ddbdisplay");
        when 4 =>          -- Show Step
              option:="2";
              put("Enter step_id: ");
              get_line(Name3,Length);
              for i in Length+1..64 loop
                  Name3(i):=' ';
              end loop;
              Show_Step(Name1,option,Name3);
              system_call("more ddbdisplay");
          when 5 =>         -- Create Prototype
                option:="1";
             put("Enter prototype's name: ");
             get_line(Name3,Length);
             for i in Length+1..64 loop
                 Name3(i):=' ';
              end loop;
              Create_Prototype(Name1,option,Name3);
                  put("More subcomponents to add[answer
                        Y/N]: ");
                get(answer);
                skip_line;
                while answer='y' or answer='Y' loop
                   put("Enter component name: ");
                   get_line(Name5,Length);
                   for i in Length+1..64 loop
                       Name5(i):=' ';
                   end loop;
                   put("Enter parent component name: ");
                   get_line(Name7,Length);
                   for i in Length+1..64 loop
                       Name7(i):=' ';
                   end loop;
                   Add_SubComponent(Name1,"4",Name3,
                           "1:1",Name5,"1:1",Name7);
                  put("More subcomponents to add[answer
                        Y/N]: ");
                   get(answer);
                   skip_line;
                end loop;
    when 6 =>           -- create substep
```

```
        option:="b";
        put("Enter parent step_id: ");
        get_line(Name3,Length);
        for i in Length+1..64 loop
            Name3(i):=' ';
        end loop;
        put("Enter primary input: ");
        get_line(Name4,Length);
        for i in Length+1..64 loop
            Name4(i):=' ';
        end loop;
        put("Enter Estimated duration: ");
        get_line(Name5,Length);
        for i in Length+1..64 loop
            Name5(i):=' ';
        end loop;
        create_substep(Name1,option,Name3,
                    Name4,Name5);
        put_designers(Name1,"2");
        get_sched_data(Name1,"0");
        system_call("cat temps temp8 > temp9");
        system_call("rm temps temp8");
        system_call("mv temp9 temps");
        get_sched_data_1(Name1,"0");
        main(0);
        Show_Schedule(Name1,"2");
        system_call("more ddbdisplay");
    when 7 =>       -- commit substep
        begin
        option:="d";
        put("Enter step_id: ");
        get_line(Name3,Length);
        for i in Length+1..64 loop
            Name3(i):=' ';
        end loop;
        Show_Step(Name1,option,Name3);
        get_current_time(the_time);
       remove_step_from_schedule(Name1,"f",Name3,
                                          the_time);
        OPEN(data_file1, IN_FILE,"temp4");
        get_line(data_file1,an_id,length);
          for i in Length+1..5 loop
              an_id(i):=' ';
          end loop;
            if an_id(1) ='N' then
        WHILE not END_OF_FILE(data_file1) loop
```

```
            get_line(data_file1,an_id,length);
            for i in Length+1..5 loop
                an_id(i):=' ';
            end loop;
            get_line(data_file1,Name4,length);
            for i in Length+1..64 loop
                Name4(i):=' ';
             end loop;
            update_time(Name1,"5",an_id,the_time);
            auto_mail(Name4,an_id);
            Change_status(Name1,"5", Name4);
            general_update(Name1,"7",an_id,"3");
          end loop;
          elsif an_id(1) ='R' then
          put_designers(Name1,"2");
          get_sched_data(Name1,"0");
          get_sched_data_1(Name1,"0");
          main(0);
          Show_Schedule(Name1,"2");
          system_call("more ddbdisplay");
          end if;
          CLOSE(data_file1);
       exception
          when NAME_ERROR=>
          put_line("No ready assignment for this
                    designer");
        end;
    when 8 =>          -- dump version
       option:="9";    -- option=6  for dump compo.
          put("Enter prototype's name: ");
          get_line(Name3,Length);
          for i in Length+1..64 loop
              Name3(i):=' ';
          end loop;
          put("Enter component name: ");
          get_line(Name5,Length);
          for i in Length+1..64 loop
              Name5(i):=' ';
          end loop;
          put("Enter variation and version: ");
          get_line(Name6,Length);
          for i in Length+1..5 loop
              Name6(i):=' ';
          end loop;
      Dump_version(Name1,option,Name3,Name5,Name6);
    when 9 =>    -- dump spec file
```

```
        option:="a";
        put("Enter prototype's name: ");
        get_line(Name3,Length);
        for i in Length+1..64 loop
             Name3(i):=' ';
        end loop;
        put("Enter component name: ");
        get_line(Name5,Length);
        for i in Length+1..64 loop
            Name5(i):=' ';
        end loop;
        put("Enter variation and version: ");
        get_line(Name6,Length);
        for i in Length+1..5 loop
            Name6(i):=' ';
        end loop;
    Dump_version(Name1,option,Name3,Name5,Name6);
  when 10 =>      -- dump Imp. file
        option:="b";
        put("Enter prototype's name: ");
        get_line(Name3,Length);
        for i in Length+1..64 loop
             Name3(i):=' ';
        end loop;
        put("Enter component name: ");
        get_line(Name5,Length);
        for i in Length+1..64 loop
            Name5(i):=' ';
        end loop;
        put("Enter variation and version: ");
        get_line(Name6,Length);
        for i in Length+1..5 loop
            Name6(i):=' ';
        end loop;
    Dump_version(Name1,option,Name3,Name5,Name6);
   when 11 =>       -- show schedule
        option:="2";
        Show_Schedule(Name1,option);
        system_call("more ddbdisplay");
    when 12 =>
         put("thank you .... Bye ...Bye");
      new_line ;
     exit;
   -- exception handling for selector case.
   when others =>
```

```
                put(" BAD CHOICE. PLEASE TRY AGAIN");
                new_line ;
            end case;
          end loop;

end  DESIGNER_INTERFACE;
```

ddb_Interface.a (Tae for manager interface) ******************************

```
-- Title              : Tae program for manager interface
-- Author             : Salah badr
-- Date               : 25 October 1993
-- Revised            :
-- System             : Suns7
-- Compiler           : VerdixAda
-- Description        :


with tae; use tae;
with X_Windows;
with text_io; use text_io;
with ecs_manager; use ecs_manager;

procedure ddb_interface is

-- FILE: ddb_interface_support_spec.a
-- Supporting procedures for ddb_interface
--   Including event handling routines.

package ddb_interface_support is


    package taefloat_io is new text_io.float_io (taefloat);
    package taeint_io is new text_io.integer_io(taeint);

    package int_io is new text_io.integer_io(integer); use
          int_io;

    procedure initializePanels (file : in string);  -- NOTE:
          params changed

    procedure CLEAR_STEP_INFO;


    -- BEGIN EVENT_HANDLERs
```

```
procedure main_selection_1 (info : in
      tae_wpt.event_context_ptr);
procedure editstep_base_version (info : in
      tae_wpt.event_context_ptr);
procedure editstep_pri_input (info : in
      tae_wpt.event_context_ptr);
procedure editstep_predecessors (info : in
      tae_wpt.event_context_ptr);
procedure editstep_priority (info : in
      tae_wpt.event_context_ptr);
procedure editstep_exp_level (info : in
      tae_wpt.event_context_ptr);
procedure editstep_deadline (info : in
      tae_wpt.event_context_ptr);
procedure editstep_est_duration (info : in
      tae_wpt.event_context_ptr);
procedure editstep_sec_input (info : in
      tae_wpt.event_context_ptr);
procedure editstep_affected (info : in
      tae_wpt.event_context_ptr);
procedure editstep_return (info : in
      tae_wpt.event_context_ptr);
procedure editstep_apply_step (info : in
      tae_wpt.event_context_ptr);
procedure editstep_cancel_step (info : in
      tae_wpt.event_context_ptr);
procedure editteam_name (info : in
      tae_wpt.event_context_ptr);
procedure editteam_ex_opt (info : in
      tae_wpt.event_context_ptr);
procedure editteam_d_cancel (info : in
      tae_wpt.event_context_ptr);
procedure editteam_designers (info : in
      tae_wpt.event_context_ptr);
procedure editteam_selection_3 (info : in
      tae_wpt.event_context_ptr);
procedure confirm_yes (info : in
      tae_wpt.event_context_ptr);
procedure confirm_no (info : in
      tae_wpt.event_context_ptr);
procedure s_select_s_select_item (info : in
      tae_wpt.event_context_ptr);
procedure showstep_s_select_item (info : in
      tae_wpt.event_context_ptr);
procedure show_step_number (info : in
      tae_wpt.event_context_ptr);
```

```
    procedure show_show_finish (info : in
          tae_wpt.event_context_ptr);
    procedure text1_display_done (info : in
          tae_wpt.event_context_ptr);
    procedure text1_text_item_1 (info : in
          tae_wpt.event_context_ptr);
    procedure steptype_type_selection (info : in
          tae_wpt.event_context_ptr);
    procedure editnum_s_select_item (info : in
          tae_wpt.event_context_ptr);
    -- END EVENT_HANDLERs

end ddb_interface_support;

-- ENDFILE: ddb_interface_support_spec.a

-------------------------------------------------------------

    use ddb_interface_support;
    use tae.tae_misc;

    theDisplay : X_Windows.Display;
    user_ptr : tae_wpt.event_context_ptr;
    main_info : tae_wpt.event_context_ptr;
    editstep_info : tae_wpt.event_context_ptr;
    editteam_info : tae_wpt.event_context_ptr;
    confirm_info : tae_wpt.event_context_ptr;
    s_select_info : tae_wpt.event_context_ptr;
    showstep_info : tae_wpt.event_context_ptr;
    show_info : tae_wpt.event_context_ptr;
    text1_info : tae_wpt.event_context_ptr;
    steptype_info : tae_wpt.event_context_ptr;
    editnum_info : tae_wpt.event_context_ptr;
    etype : wpt_eventtype;
    wptEvent : tae_wpt.wpt_eventptr;

   dummy : boolean;   -- used to clear out the wpt event queue

    type activity_selector is (editing, creating);

    MAX_DESIGNERS    : integer := 20;
    MAX_SEC_INPUTS   : integer := 20;

   secondary_inputs : s_vector(1..MAX_SEC_INPUTS) := (others
         => new string(1..64));
   affected_modules : s_vector(1..MAX_SEC_INPUTS) := (others
```

```
        => new string(1..64));
    designer_info    : s_vector(1..MAX_DESIGNERS) := (others
        => new string(1..64));
    deadline         : string(1..24);
    designer         : string(1..24);
    start_time       : string(1..24);
    status           : string(1..24);
    finish_time      : string(1..24);
    sub_steps        : string(1..24);
    predecessors     : string(1..24);
    expertise_level  : string(1..24);
    designer_status  : string(1..24);
    base_version     : string(1..64);
    primary_input    : string(1..64);

    data_file        : text_io.file_type;
    length           : integer;
    counter          : integer;

    temp_string      : string(1..64);

    priority         : integer;
    est_duration     : integer;
    step_number      : integer;

    editing_or_creating  :  activity_selector;

-------------------------------------------------------------


-- FILE: ddb_interface_support_body.a

package body ddb_interface_support is

procedure initializePanels (file : in string) is

          use tae.tae_co;
          use tae.tae_misc;

          tmp_info : tae_wpt.event_context_ptr;
        dummy    : BOOLEAN;

    begin

          -- do one Co_New and Co_ReadFile per resource file
          tmp_info := new tae_wpt.event_context;
```

```
    Co_New (0, tmp_info.collection);
    -- could pass P_ABORT if you prefer
    Co_ReadFile (tmp_info.collection, file, P_CONT);

-- pair of Co_Finds for each panel in this resource file

  main_info := new tae_wpt.event_context;
  main_info.collection := tmp_info.collection;
  Co_Find (main_info.collection, "main_v",
    main_info.view);
  Co_Find (main_info.collection, "main_t",
    main_info.target);

  editstep_info := new tae_wpt.event_context;
  editstep_info.collection := tmp_info.collection;
  Co_Find (editstep_info.collection, "editstep_v",
    editstep_info.view);
  Co_Find (editstep_info.collection, "editstep_t",
    editstep_info.target);

  editteam_info := new tae_wpt.event_context;
  editteam_info.collection := tmp_info.collection;
  Co_Find (editteam_info.collection, "editteam_v",
    editteam_info.view);
  Co_Find (editteam_info.collection, "editteam_t",
    editteam_info.target);

  confirm_info := new tae_wpt.event_context;
  confirm_info.collection := tmp_info.collection;
  Co_Find (confirm_info.collection, "confirm_v",
    confirm_info.view);
  Co_Find (confirm_info.collection, "confirm_t",
    confirm_info.target);

  s_select_info := new tae_wpt.event_context;
  s_select_info.collection := tmp_info.collection;
  Co_Find (s_select_info.collection, "s_select_v",
    s_select_info.view);
  Co_Find (s_select_info.collection, "s_select_t",
    s_select_info.target);

  showstep_info := new tae_wpt.event_context;
  showstep_info.collection := tmp_info.collection;
  Co_Find (showstep_info.collection, "showstep_v",
    showstep_info.view);
  Co_Find (showstep_info.collection, "showstep_t",
```

```
    showstep_info.target);

  show_info := new tae_wpt.event_context;
  show_info.collection := tmp_info.collection;
  Co_Find (show_info.collection, "show_v",
    show_info.view);
  Co_Find (show_info.collection, "show_t",
    show_info.target);

  text1_info := new tae_wpt.event_context;
  text1_info.collection := tmp_info.collection;
  Co_Find (text1_info.collection, "text1_v",
    text1_info.view);
  Co_Find (text1_info.collection, "text1_t",
    text1_info.target);

  steptype_info := new tae_wpt.event_context;
  steptype_info.collection := tmp_info.collection;
  Co_Find (steptype_info.collection, "steptype_v",
    steptype_info.view);
  Co_Find (steptype_info.collection, "steptype_t",
    steptype_info.target);

  editnum_info := new tae_wpt.event_context;
  editnum_info.collection := tmp_info.collection;
  Co_Find (editnum_info.collection, "editnum_v",
    editnum_info.view);
  Co_Find (editnum_info.collection, "editnum_t",
    editnum_info.target);


-- Since there can now be MULTIPLE INITIAL PANELS defined
    from
 -- within the TAE WorkBench, call Wpt_NewPanel for each
    panel
  -- defined to be an initial panel (but not usually all
    the panels
   -- which appear in the resource file).


  if main_info.panel_id = NULL_PANEL_ID then
      tae_wpt.Wpt_NewPanel ("", main_info.target,
    main_info.view,
         X_Windows.Null_Window, main_info,
    tae_wpt.WPT_PREFERRED,
         main_info.panel_id);
```

```
        else
            tae_wpt.Wpt_SetPanelState (
               main_info.panel_id, tae_wpt.WPT_PREFERRED);
        end if;

        dummy := Tae_Wpt.Wpt_Pending;

     end initializePanels;

procedure CLEAR_STEP_INFO is

   begin

    for i in 1..20 loop
      secondary_inputs(i).all :=
         "                                                    ";
      affected_modules(i).all :=
         "                                                    ";
    end loop;

    deadline          := "                          ";
    designer          := "                          ";
    start_time        := "                          ";
    status            := "                          ";
    finish_time       := "                          ";
    sub_steps         := "                          ";
    predecessors      := "                          ";
    expertise_level   := "                          ";
    base_version      :=
          "
          ";
    primary_input     :=
          "
          ";

    temp_string       :=
          "
          ";

    priority          := 0;
    est_duration      := 0;

end CLEAR_STEP_INFO;

--
--        BEGIN EVENT_HANDLERs
```

```
--

procedure main_selection_1 (info : in
          tae_wpt.event_context_ptr) is
    value : array (1..1) of string
          (1..tae_taeconf.STRINGSIZE);
    count : taeint;

        begin
      text_io.put ("Panel main, parm selection_1: value = ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
        else
        tae_vm.Vm_Extract_SVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (value(1));
        end if;
        if (FALSE) then null;
        elsif s_equal (value(1), "show prototypes") then
SELECTOR := 1;
ECS1;
           if text1_info.panel_id = NULL_PANEL_ID then
              tae_wpt.Wpt_NewPanel ("", text1_info.target,
          text1_info.view,
                   X_Windows.Null_Window, text1_info,
          tae_wpt.WPT_PREFERRED,
                   text1_info.panel_id);
           else
               tae_wpt.Wpt_SetPanelState (
               text1_info.panel_id, tae_wpt.WPT_PREFERRED);
           end if;
        elsif s_equal (value(1), "show steps") then null;
           if steptype_info.panel_id = NULL_PANEL_ID then
            tae_wpt.Wpt_NewPanel ("", steptype_info.target,
          steptype_info.view,
                   X_Windows.Null_Window, steptype_info,
          tae_wpt.WPT_PREFERRED,
                   steptype_info.panel_id);
           else
               tae_wpt.Wpt_SetPanelState (
                   steptype_info.panel_id,
          tae_wpt.WPT_PREFERRED);
           end if;
        elsif s_equal (value(1), "show step details") then
          null;
```

```
            if showstep_info.panel_id = NULL_PANEL_ID then
               tae_wpt.Wpt_NewPanel ("", showstep_info.target,
          showstep_info.view,
                    X_Windows.Null_Window, showstep_info,
          tae_wpt.WPT_PREFERRED,
                    showstep_info.panel_id);
            else
                tae_wpt.Wpt_SetPanelState (
                    showstep_info.panel_id,
          tae_wpt.WPT_PREFERRED);
            end if;

          Tae_Wpt.Wpt_SetIntg(showstep_info.panel_id,"s_sele
          ct_item",Taeint(0));

        elsif s_equal (value(1), "show schedule") then
SELECTOR := 4;
ECS1;
            if text1_info.panel_id = NULL_PANEL_ID then
               tae_wpt.Wpt_NewPanel ("", text1_info.target,
          text1_info.view,
                    X_Windows.Null_Window, text1_info,
          tae_wpt.WPT_PREFERRED,
                    text1_info.panel_id);
            else
                tae_wpt.Wpt_SetPanelState (
                text1_info.panel_id, tae_wpt.WPT_PREFERRED);
            end if;
       elsif s_equal (value(1), "create prototype") then null;
         elsif s_equal (value(1), "create step") then

editing_or_creating := creating;
CLEAR_STEP_INFO;

            if editstep_info.panel_id = NULL_PANEL_ID then
               tae_wpt.Wpt_NewPanel ("", editstep_info.target,
          editstep_info.view,
                    X_Windows.Null_Window, editstep_info,
          tae_wpt.WPT_PREFERRED,
                    editstep_info.panel_id);
            else
                tae_wpt.Wpt_SetPanelState (
                    editstep_info.panel_id,
          tae_wpt.WPT_PREFERRED);
            end if;
```

```
-- still need to clear displayed affected modules and
        secondary inputs


        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"base
        _version",
                                            base_version);

        Tae_Wpt.Wpt_SetIntg(editstep_info.panel_id,"est_du
        ration",
                                        Taeint(est_duration));

        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"exp_
        level",
                                            expertise_level);
    Tae_Wpt.Wpt_SetIntg(editstep_info.panel_id,"priority",
                                          Taeint(priority));

        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"desi
        gner",
                                            designer);

        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"dead
        line",
                                            deadline);

        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"star
        t_time",
                                            start_time);

        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"fini
        sh_time",
                                            finish_time);

        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"pri_
        input",
                                            primary_input);

        Tae_Wpt.Wpt_SetIntg(editstep_info.panel_id,"step_n
        umber",
                                            TaeInt(0));

        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"sub_
        steps",
                                            sub_steps);
```

```
            Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"pred
            ecessors",
                                                predecessors);


          elsif s_equal (value(1), "edit step =>") then null;
              if editnum_info.panel_id = NULL_PANEL_ID then
                tae_wpt.Wpt_NewPanel ("", editnum_info.target,
            editnum_info.view,
                      X_Windows.Null_Window, editnum_info,
            tae_wpt.WPT_PREFERRED,
                      editnum_info.panel_id);
              else
                  tae_wpt.Wpt_SetPanelState (
                editnum_info.panel_id, tae_wpt.WPT_PREFERRED);
              end if;

            Tae_Wpt.Wpt_SetIntg(editnum_info.panel_id,"s_selec
            t_item",Taeint(0));

          elsif s_equal (value(1), "edit team =>") then null;
              if editteam_info.panel_id = NULL_PANEL_ID then
               tae_wpt.Wpt_NewPanel ("", editteam_info.target,
            editteam_info.view,
                      X_Windows.Null_Window, editteam_info,
            tae_wpt.WPT_PREFERRED,
                      editteam_info.panel_id);
              else
                  tae_wpt.Wpt_SetPanelState (
                      editteam_info.panel_id,
            tae_wpt.WPT_PREFERRED);
              end if;

SELECTOR := 13;  -- remove data transfer file
ECS1;
SELECTOR := 15;  -- put 'show designers' information in data
            transfer file
ECS1;

-- read in the designers from the transfer file (ddbdisplay)
            into the editteam panel

              text_io.OPEN(data_file,
            text_io.IN_FILE,"ddbdisplay");

              counter := 1;
```

```
          while not end_of_file(data_file) loop


        get_line(data_file,designer_info(counter).all,leng
        th);

        TAE_Wpt.Wpt_SetStringConstraints(editteam_info.pan
        el_id,
                        "designers", TaeInt(counter),
        designer_info);

            counter := counter + 1;

          end loop;

          text_io.CLOSE(data_file);

          for i in counter..MAX_DESIGNERS loop

             designer_info(i).all :=

        "
        ";

        TAE_Wpt.Wpt_SetStringConstraints(editteam_info.pan
        el_id,
                        "designers", TaeInt(counter),
        designer_info);

          end loop;

      elsif s_equal (value(1), "approve step") then
Selector := 8;
          if s_select_info.panel_id = NULL_PANEL_ID then
           tae_wpt.Wpt_NewPanel ("", s_select_info.target,
        s_select_info.view,
                  X_Windows.Null_Window, s_select_info,
        tae_wpt.WPT_PREFERRED,
                  s_select_info.panel_id);
          else
              tae_wpt.Wpt_SetPanelState (
                  s_select_info.panel_id,
        tae_wpt.WPT_PREFERRED);
          end if;
```

```
            Tae_Wpt.Wpt_SetIntg(s_select_info.panel_id,"s_sele
            ct_item",Taeint(0));

          elsif s_equal (value(1), "schedule step") then
Selector := 9;
            if s_select_info.panel_id = NULL_PANEL_ID then
             tae_wpt.Wpt_NewPanel ("", s_select_info.target,
            s_select_info.view,
                     X_Windows.Null_Window, s_select_info,
            tae_wpt.WPT_PREFERRED,
                     s_select_info.panel_id);
            else
                tae_wpt.Wpt_SetPanelState (
                   s_select_info.panel_id,
            tae_wpt.WPT_PREFERRED);
            end if;

            Tae_Wpt.Wpt_SetIntg(s_select_info.panel_id,"s_sele
            ct_item",Taeint(0));

          elsif s_equal (value(1), "commit step") then
SELECTOR := 10;
            if s_select_info.panel_id = NULL_PANEL_ID then
             tae_wpt.Wpt_NewPanel ("", s_select_info.target,
            s_select_info.view,
                     X_Windows.Null_Window, s_select_info,
            tae_wpt.WPT_PREFERRED,
                     s_select_info.panel_id);
            else
                tae_wpt.Wpt_SetPanelState (
                   s_select_info.panel_id,
            tae_wpt.WPT_PREFERRED);
            end if;

            Tae_Wpt.Wpt_SetIntg(s_select_info.panel_id,"s_sele
            ct_item",Taeint(0));

          elsif s_equal (value(1), "suspend step") then
SELECTOR := 11;
            if s_select_info.panel_id = NULL_PANEL_ID then
             tae_wpt.Wpt_NewPanel ("", s_select_info.target,
            s_select_info.view,
                     X_Windows.Null_Window, s_select_info,
            tae_wpt.WPT_PREFERRED,
                     s_select_info.panel_id);
            else
```

```
                tae_wpt.Wpt_SetPanelState (
                   s_select_info.panel_id,
          tae_wpt.WPT_PREFERRED);
            end if;

          Tae_Wpt.Wpt_SetIntg(s_select_info.panel_id,"s_sele
          ct_item",Taeint(0));

        elsif s_equal (value(1), "abandon step") then
SELECTOR := 12;
            if s_select_info.panel_id = NULL_PANEL_ID then
             tae_wpt.Wpt_NewPanel ("", s_select_info.target,
          s_select_info.view,
                   X_Windows.Null_Window, s_select_info,
          tae_wpt.WPT_PREFERRED,
                   s_select_info.panel_id);
            else
                tae_wpt.Wpt_SetPanelState (
                   s_select_info.panel_id,
          tae_wpt.WPT_PREFERRED);
            end if;

          Tae_Wpt.Wpt_SetIntg(s_select_info.panel_id,"s_sele
          ct_item",Taeint(0));

        elsif s_equal (value(1), "quit") then null;
        end if;

        tae_wpt.Wpt_PanelReset(main_info.panel_id);

    end main_selection_1;

procedure editstep_base_version (info : in
          tae_wpt.event_context_ptr) is
    value : array (1..1) of string
          (1..tae_taeconf.STRINGSIZE);
    count : taeint;

        begin
      text_io.put ("Panel editstep, parm base_version: value
          = ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
        else
        tae_vm.Vm_Extract_SVAL (info.parm_ptr, 1, value(1));
```

```
            text_io.put_line (value(1));
         end if;
base_version(1..64) := value(1)(1..64);
    end editstep_base_version;

procedure editstep_pri_input (info : in
          tae_wpt.event_context_ptr) is
    value : array (1..1) of string
          (1..tae_taeconf.STRINGSIZE);
    count : taeint;

        begin
       text_io.put ("Panel editstep, parm pri_input: value =
          ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
        else
        tae_vm.Vm_Extract_SVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (value(1));
        end if;
primary_input(1..64) := value(1)(1..64);
    end editstep_pri_input;

procedure editstep_predecessors (info : in
          tae_wpt.event_context_ptr) is
    value : array (1..1) of string
          (1..tae_taeconf.STRINGSIZE);
    count : taeint;

        begin
      text_io.put ("Panel editstep, parm predecessors: value
          = ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
        else
        tae_vm.Vm_Extract_SVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (value(1));
        end if;

-- need to turn the predecessors string into suitable numbers
          for entry
-- into the ddb
predecessors(1..24) := value(1)(1..24);
```

```
-- currently only ADD PREDECESSOR is supported.


    end editstep_predecessors;

procedure editstep_priority (info : in
          tae_wpt.event_context_ptr) is
    value : array (1..1) of taeint;
    count : taeint;

        begin
        text_io.put ("Panel editstep, parm priority: value =
          ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
        else
        tae_vm.Vm_Extract_IVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (taeint'image(value(1)));
        end if;
-- assign priority := value(1) -- note type incompatability
        priority := integer(value(1));

    end editstep_priority;

procedure editstep_exp_level (info : in
          tae_wpt.event_context_ptr) is
    value : array (1..1) of string
          (1..tae_taeconf.STRINGSIZE);
    count : taeint;

        begin
       text_io.put ("Panel editstep, parm exp_level: value =
          ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
        else
        tae_vm.Vm_Extract_SVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (value(1));
        end if;
expertise_level(1..6) := value(1)(1..6);
    end editstep_exp_level;

procedure editstep_deadline (info : in
```

```
            tae_wpt.event_context_ptr) is
        value : array (1..1) of string
            (1..tae_taeconf.STRINGSIZE);
        count : taeint;

        begin
        text_io.put ("Panel editstep, parm deadline: value =
          ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
        else
        tae_vm.Vm_Extract_SVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (value(1));
        end if;
deadline(1..24) := value(1)(1..24);
    end editstep_deadline;

procedure editstep_est_duration (info : in
            tae_wpt.event_context_ptr) is
    value : array (1..1) of taeint;
    count : taeint;

        begin
      text_io.put ("Panel editstep, parm est_duration: value
          = ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
        else
        tae_vm.Vm_Extract_IVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (taeint'image(value(1)));
        end if;

--  assign estimated_duration := value(1) -- note type
          incompatability
        est_duration := integer(value(1));

    end editstep_est_duration;

procedure editstep_sec_input (info : in
            tae_wpt.event_context_ptr) is
    value : array (1..1) of string
            (1..tae_taeconf.STRINGSIZE);
    count : taeint;
```

```
        begin
       text_io.put ("Panel editstep, parm sec_input: value =
          ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
        else
        tae_vm.Vm_Extract_SVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (value(1));
        end if;
-- need to modify the secandary input array to reflect what
         is in the TAE window
    end editstep_sec_input;

procedure editstep_affected (info : in
          tae_wpt.event_context_ptr) is
    value : array (1..1) of string
          (1..tae_taeconf.STRINGSIZE);
    count : taeint;

        begin
       text_io.put ("Panel editstep, parm affected: value =
          ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
        else
        tae_vm.Vm_Extract_SVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (value(1));
-- need to modify the affected module array to reflect any new
         info
        end if;
    end editstep_affected;


procedure editstep_return (info : in
          tae_wpt.event_context_ptr) is

        begin
         if not (editstep_info.panel_id = NULL_PANEL_ID) then
              tae_wpt.Wpt_PanelErase(info.panel_id); end if;

    end editstep_return;


procedure editstep_apply_step (info : in
```

```
          tae_wpt.event_context_ptr) is
    value : array (1..1) of string
          (1..tae_taeconf.STRINGSIZE);
    count : taeint;

        begin
        text_io.put ("Panel editstep, parm apply_step: value
          = ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
        else
        tae_vm.Vm_Extract_SVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (value(1));
        end if;
-- need to write all step stuff to ddb

   if editing_or_creating = creating then  -- creating a new
         step

      begin

-- note that the parsing of the base version and the primary
         input
-- is VERY rudimentary.  We are currently expecting the user
         to know
-- the required input form:  base_version var:ver
--                           primary_input var:ver


         if base_version(1..3) /= "    " then
            if primary_input(1..3) /= "    " then

               SELECTOR := 6;
               Name3(1..64) := base_version(1..64);
               Name4(1..64) := primary_input(1..64);
               ECS1;

               put_line("step creation complete");

            else put_line("base version and primary input
         required for step creation.");
            end if;
      else put_line("base version and primary input required
         for step creation.");
        end if;
```

```
      end;

   else  -- editing a step

      begin

--------------
                    Name4(1..3):="0 0";
                    for i in 4..64 loop
                        Name4(i):=' ';
                    end loop;
                    Name5(1):='0';
                    for i in 2..64 loop
                        Name5(i):=' ';
                    end loop;
                    Name6(1..3):="0 0";
                    for i in 4..64 loop
                        Name6(i):=' ';
                    end loop;
                    Name7(1):='0';
                    for i in 2..64 loop
                        Name7(i):=' ';
                    end loop;
                    Name8(1..3):="0 0";
                    for i in 4..64 loop
                        Name8(i):=' ';
                    end loop;
                    Name9(1):='0';
                    for i in 2..64 loop
                        Name9(i):=' ';
                    end loop;
                    NameA(1):='0';
                    for i in 2..64 loop
                        NameA(i):=' ';
                    end loop;
                    NameB(1):='0';
                    for i in 2..64 loop
                        NameB(i):=' ';
                    end loop;
                    NameC(1):='0';
                    for i in 2..64 loop
                        NameC(i):=' ';
                    end loop;
                    NameD(1):='5';
                    for i in 2..64 loop
```

```
                        NameD(i):=' ';
                    end loop;
                    NameE(1..4):="00 0";
                    for i in 5..64 loop
                        NameE(i):=' ';
                    end loop;

          taeint_io.put(Name3,taeint(step_number));   -- step_id
--          if primary_input(1..3)/="   " then
--          Name4(1..64) :=primary_input(1..64);        -- for
          addition
--          end if;
--          if primary_input(1..3)/="   " then
--          Name5(1..64) :=primary_input(1..64);        -- for
          deletion
--          end if;
--          if secondary_inputs(1)(1..3)/="   " then
--          Name6(1..64) := secondary_inputs(1)(1..64);  -- for
          addition
--          end if;
--          if secondary_inputs(1)(1..3)/="   " then
--          Name7(1..64) := secondary_inputs(1)(1..64);  -- for
          deletion
--          end if;
--          if affected_modules(1)(1..3)/="   " then
--          Name8(1..64) := affected_modules(1)(1..64);  -- for
          addition
--          end if;
--          if affected_modules(1)(1..3)/="   " then
--          Name9(1..64) := affected_modules(1)(1..64);  -- for
          deletion
--          end if;
         taeint_io.put(NameA,taeint(priority));       -- priority
          if predecessors(1..3)/="   " then
          NameB(1..3) := predecessors(1..3);
          end if;
          taeint_io.put(NameC,taeint(est_duration));
          if expertise_level(1..3) = "low" then NameD(1) := '0';
          end if;
          if expertise_level(1..3) = "med" then NameD(1) := '1';
          end if;
          if expertise_level(1..3) = "hig" then NameD(1) := '2';
          end if;
          NameE(1..24) := deadline(1..24);
put("step number:"); put(Name3); new_line;
put("priority:"); put(NameA); new_line;
```

```
put("estimated duration:"); put(NameC); new_line;

          put_line("apply has been depressed while in the edit
          mode");
          put_line("calling ECS.");

SELECTOR := 7;
ECS1;

          put_line("ECS operations complete");
       end;

    end if;

    end editstep_apply_step;

procedure editstep_cancel_step (info : in
          tae_wpt.event_context_ptr) is
    value : array (1..1) of string (1..tae_taeconf.STRINGSIZE);
    count : taeint;

        begin
       text_io.put ("Panel editstep, parm cancel_step: value = ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
        else
           tae_vm.Vm_Extract_SVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (value(1));
        end if;

-- should probably retrieve all of the old step information and
          refresh
-- the edit step window

        CLEAR_STEP_INFO;

                      for i in 1..MAX_SEC_INPUTS loop

          TAE_Wpt.Wpt_SetStringConstraints(editstep_info.panel_id
          ,
                                "sec_input", TaeInt(i),
          secondary_inputs);


          TAE_Wpt.Wpt_SetStringConstraints(editstep_info.panel_id
```

```
                    "affected", TaeInt(i), affected_modules);

                end loop;


        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"base_vers
        ion",
                                        base_version);
      Tae_Wpt.Wpt_SetIntg(editstep_info.panel_id,"est_duration",
                                        Taeint(est_duration));
       Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"exp_level",
                                        expertise_level);
       Tae_Wpt.Wpt_SetIntg(editstep_info.panel_id,"priority",
                                        Taeint(priority));
       Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"designer",
                                        designer);
       Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"deadline",
                                        deadline);
      Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"start_time",
                                        start_time);
    Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"finish_time",
                                        finish_time);
       Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"pri_input",
                                        primary_input);
       Tae_Wpt.Wpt_SetIntg(editstep_info.panel_id,"step_number",
                                        taeint(step_number));
       Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"sub_steps",
                                        sub_steps);

        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"predecess
        ors",
                                        predecessors);


    end editstep_cancel_step;

procedure editteam_name (info : in tae_wpt.event_context_ptr) is
    value : array (1..1) of string (1..tae_taeconf.STRINGSIZE);
    count : taeint;

        begin
        text_io.put ("Panel editteam, parm name: value = ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
```

```
        else
           tae_vm.Vm_Extract_SVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (value(1));
        end if;
designer(1..24) := value(1)(1..24);
    end editteam_name;

procedure editteam_ex_opt (info : in tae_wpt.event_context_ptr) is
    value : array (1..1) of string (1..tae_taeconf.STRINGSIZE);
    count : taeint;

        begin
        text_io.put ("Panel editteam, parm ex_opt: value = ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
        else
           tae_vm.Vm_Extract_SVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (value(1));
        Tae_Wpt.Wpt_SetString(editteam_info.panel_id,"expertise",
                                              value(1));
expertise_level(1..24) := value(1)(1..24);
        end if;
    end editteam_ex_opt;

procedure editteam_d_cancel (info : in tae_wpt.event_context_ptr)
          is
    value : array (1..1) of string (1..tae_taeconf.STRINGSIZE);
    count : taeint;

        begin
        text_io.put ("Panel editteam, parm d_cancel: value = ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
        else
           tae_vm.Vm_Extract_SVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (value(1));
        end if;
        if info.panel_id = NULL_PANEL_ID then
            tae_wpt.Wpt_NewPanel ("", info.target, info.view,
             X_Windows.Null_Window, info, tae_wpt.WPT_PREFERRED,
               info.panel_id);
        else
            tae_wpt.Wpt_SetPanelState (
               info.panel_id, tae_wpt.WPT_PREFERRED);
```

```
        end if;
    Tae_Wpt.Wpt_SetString(editteam_info.panel_id,"name",
                          "                              ");
    Tae_Wpt.Wpt_SetString(editteam_info.panel_id,"expertise",
                          "       ");
    Tae_Wpt.Wpt_SetString(editteam_info.panel_id,"status",
                          "         ");
    end editteam_d_cancel;

procedure editteam_designers (info : in tae_wpt.event_context_ptr)
          is
    value : array (1..1) of string (1..tae_taeconf.STRINGSIZE);
    count : taeint;

        begin
        text_io.put ("Panel editteam, parm designers: value = ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
        else
           tae_vm.Vm_Extract_SVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (value(1));
        end if;
    Tae_Wpt.Wpt_SetString(editteam_info.panel_id,"name",
                          value(1)(1..24));
designer(1..24) := value(1)(1..24);
    Tae_Wpt.Wpt_SetString(editteam_info.panel_id,"expertise",
                          value(1)(25..30));
expertise_level(1..6) := value(1)(25..30);
    Tae_Wpt.Wpt_SetString(editteam_info.panel_id,"status",
                          value(1)(44..51));
designer_status(1..4) := value(1)(44..47);

    end editteam_designers;

procedure editteam_selection_3 (info : in
          tae_wpt.event_context_ptr) is
    value : array (1..1) of string (1..tae_taeconf.STRINGSIZE);
    count : taeint;

        begin

       text_io.put ("Panel editteam, parm selection_3: value = ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
```

```
        else
           tae_vm.Vm_Extract_SVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (value(1));
        end if;
        if (FALSE) then null;
        elsif s_equal (value(1), "add designer") then
-- add designer to ddb
          Name3(1..24) := designer(1..24);
          if expertise_level(1..3) = "low" then Name4(1) := '0';
          end if;
          if expertise_level(1..3) = "med" then Name4(1) := '1';
          end if;
          if expertise_level(1..3) = "hig" then Name4(1) := '2';
          end if;

put_line("calling ECS");

          SELECTOR := 14;
          ECS1;

          tae_wpt.Wpt_PanelReset(editteam_info.panel_id);

-- now read the new transfer file and update the TAE item

SELECTOR := 13;  -- remove transfer file
ECS1;
SELECTOR := 15;  -- write designer list to transfer file
ECS1;

put_line("designer addition complete");

-- read in the designers from the transfer file (ddbdisplay) into
          the editteam panel

           text_io.OPEN(data_file, text_io.IN_FILE,"ddbdisplay");

            counter := 1;

            while not end_of_file(data_file) loop


          get_line(data_file,designer_info(counter).all,length);

          TAE_Wpt.Wpt_SetStringConstraints(editteam_info.panel_id
          ,
                     "designers", TaeInt(counter), designer_info);
```

```
            counter := counter + 1;

          end loop;

          text_io.CLOSE(data_file);

          for i in counter..MAX_DESIGNERS loop

            designer_info(i).all :=
           "                                                    ";

        TAE_Wpt.Wpt_SetStringConstraints(editteam_info.panel_id
        ,
                  "designers", TaeInt(counter), designer_info);

          end loop;


      elsif s_equal (value(1), "delete designer") then
Name3(1..24) := designer(1..24);
Name4(1..4) :=  designer_status(1..4);
          if confirm_info.panel_id = NULL_PANEL_ID then
              tae_wpt.Wpt_NewPanel ("", confirm_info.target,
        confirm_info.view,
                  X_Windows.Null_Window, confirm_info,
        tae_wpt.WPT_PREFERRED,
                  confirm_info.panel_id);
          else
              tae_wpt.Wpt_SetPanelState (
                 confirm_info.panel_id, tae_wpt.WPT_PREFERRED);
          end if;

      elsif s_equal (value(1), "change expertise level") then

-- update designer info in ddb

          begin

            NAME3(1..24) := designer(1..24);
            if expertise_level(1..3) = "low" then Name4(1) :=
        '0'; end if;
            if expertise_level(1..3) = "med" then Name4(1) :=
        '1'; end if;
            if expertise_level(1..3) = "hig" then Name4(1) :=
        '2'; end if;
```

```
put_line("calling ECS");

              SELECTOR := 17;
              ECS1;

            tae_wpt.Wpt_PanelReset(editteam_info.panel_id);

-- now read the new transfer file and update the TAE item

SELECTOR := 13;  -- remove transfer file
ECS1;
SELECTOR := 15;  -- write designer list to transfer file
ECS1;

put_line("designer expertise modification complete");

-- read in the designers from the transfer file (ddbdisplay) into
          the editteam panel

           text_io.OPEN(data_file, text_io.IN_FILE,"ddbdisplay");

            counter := 1;

            while not end_of_file(data_file) loop


          get_line(data_file,designer_info(counter).all,length);

          TAE_Wpt.Wpt_SetStringConstraints(editteam_info.panel_id
          ,
                    "designers", TaeInt(counter), designer_info);
               counter := counter + 1;

            end loop;

            text_io.CLOSE(data_file);

            for i in counter..MAX_DESIGNERS loop

               designer_info(i).all :=
             "                                                    ";

          TAE_Wpt.Wpt_SetStringConstraints(editteam_info.panel_id
          ,
                    "designers", TaeInt(counter), designer_info);
```

```
            end loop;

            end;

        elsif s_equal (value(1), "return to main menu") then

            tae_wpt.Wpt_PanelReset(editteam_info.panel_id);

            if not (editteam_info.panel_id = NULL_PANEL_ID) then
               tae_wpt.Wpt_PanelErase(info.panel_id);  end if;
        end if;

    end editteam_selection_3;

procedure confirm_yes (info : in tae_wpt.event_context_ptr) is
    value : array (1..1) of string (1..tae_taeconf.STRINGSIZE);
    count : taeint;

        begin
        text_io.put ("Panel confirm, parm yes: value = ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
        else
           tae_vm.Vm_Extract_SVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (value(1));
        end if;
        tae_wpt.Wpt_PanelErase(info.panel_id);
-- remove designer from ddb

put_line("calling ECS");

SELECTOR := 16;
ECS1;


    tae_wpt.Wpt_PanelReset(editteam_info.panel_id);

-- clear the TAE panel items

    Tae_Wpt.Wpt_SetString(editteam_info.panel_id,"name",
                          "                              ");
    Tae_Wpt.Wpt_SetString(editteam_info.panel_id,"expertise",
                          "       ");
    Tae_Wpt.Wpt_SetString(editteam_info.panel_id,"status",
                          "         ");
```

```
-- read the new designer list from the transfer file

SELECTOR := 13;
ECS1;
SELECTOR := 15;
ECS1;

put_line("designer deletion complete");

-- read in the designers from the transfer file (ddbdisplay) into
          the editteam panel

           text_io.OPEN(data_file, text_io.IN_FILE,"ddbdisplay");

            counter := 1;

            while not end_of_file(data_file) loop


          get_line(data_file,designer_info(counter).all,length);

          TAE_Wpt.Wpt_SetStringConstraints(editteam_info.panel_id
          ,
                    "designers", TaeInt(counter), designer_info);
              counter := counter + 1;

            end loop;

            text_io.CLOSE(data_file);

            for i in counter..MAX_DESIGNERS loop

               designer_info(i).all :=
              "                                                    ";

          TAE_Wpt.Wpt_SetStringConstraints(editteam_info.panel_id
          ,
                    "designers", TaeInt(counter), designer_info);

            end loop;

   end confirm_yes;


procedure confirm_no (info : in tae_wpt.event_context_ptr) is
```

```
        begin

put_line("cancelling designer deletion");

        -- do nothing
        tae_wpt.Wpt_PanelErase(info.panel_id);

    end confirm_no;

procedure s_select_s_select_item (info : in
          tae_wpt.event_context_ptr) is
    value : array (1..1) of taeint;
    count : taeint;

        begin
        text_io.put ("Panel s_select, parm s_select_item: value =
          ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
        else
           tae_vm.Vm_Extract_IVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (taeint'image(value(1)));
taeint_io.put(Name3,(value(1)));

        end if;

put_line("calling ECS");

ECS1;

put_line("ECS operation complete");

        tae_wpt.Wpt_PanelErase(info.panel_id);
    end s_select_s_select_item;

procedure showstep_s_select_item (info : in
          tae_wpt.event_context_ptr) is
    value : array (1..1) of taeint;
    count : taeint;

        begin
        text_io.put ("Panel showstep, parm s_select_item: value =
          ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
```

```
        if count <= 0 then
           text_io.put_line ("none");
        else
           tae_vm.Vm_Extract_IVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (taeint'image(value(1)));
        end if;

taeint_io.put(Name3,(value(1)));
step_number := integer(value(1));
SELECTOR := 3;
ECS1;  -- this creates a file called 'ddbdisplay' and puts all the
          step
       -- info in the file
CLEAR_STEP_INFO;

      --  now create the new window

        tae_wpt.Wpt_PanelErase(showstep_info.panel_id);
        if show_info.panel_id = NULL_PANEL_ID then
            tae_wpt.Wpt_NewPanel ("", show_info.target,
          show_info.view,
               X_Windows.Null_Window, show_info,
          tae_wpt.WPT_PREFERRED,
               show_info.panel_id);
        else
            tae_wpt.Wpt_SetPanelState (
               show_info.panel_id, tae_wpt.WPT_PREFERRED);
        end if;

-- make sure that the panel items are cleared

                    for i in 1..MAX_SEC_INPUTS loop

          TAE_Wpt.Wpt_SetStringConstraints(show_info.panel_id,
                                "sec_input", TaeInt(i),
          secondary_inputs);


          TAE_Wpt.Wpt_SetStringConstraints(show_info.panel_id,
                        "affected", TaeInt(i), affected_modules);

                    end loop;
        Tae_Wpt.Wpt_SetString(show_info.panel_id,"base_version",
                                          base_version);
        Tae_Wpt.Wpt_SetIntg(show_info.panel_id,"est_duration",
                                          Taeint(est_duration));
```

```
        Tae_Wpt.Wpt_SetString(show_info.panel_id,"exp_level",
                                              expertise_level);
        Tae_Wpt.Wpt_SetIntg(show_info.panel_id,"priority",
                                              Taeint(priority));
        Tae_Wpt.Wpt_SetString(show_info.panel_id,"designer",
                                              designer);
        Tae_Wpt.Wpt_SetString(show_info.panel_id,"deadline",
                                              deadline);
        Tae_Wpt.Wpt_SetString(show_info.panel_id,"start_time",
                                              start_time);
        Tae_Wpt.Wpt_SetString(show_info.panel_id,"finish_time",
                                              finish_time);
        Tae_Wpt.Wpt_SetString(show_info.panel_id,"pri_input",
                                              primary_input);
        Tae_Wpt.Wpt_SetIntg(show_info.panel_id,"step_number",
                                              taeint(step_number));
        Tae_Wpt.Wpt_SetString(show_info.panel_id,"sub_steps",
                                              sub_steps);
        Tae_Wpt.Wpt_SetString(show_info.panel_id,"predecessors",
                                              predecessors);

-- read from file created by ECS1 into TAE variables
-- for display in 'show' panel.

text_io.put_line("opening file");

                    text_io.OPEN(data_file,
          text_io.IN_FILE,"ddbdisplay");

text_io.put_line("reading from file");

                text_io.get_line(data_file,base_version,length);
                    for i in length+1..64 loop
                       base_version(i):=' ';
                    end loop;
                    get(data_file,est_duration);
                    skip_line(data_file);
                    get(data_file,priority);
                    skip_line(data_file);

          text_io.get_line(data_file,expertise_level,length);
                    for i in length+1..24 loop
                       expertise_level(i):=' ';
                    end loop;
                    text_io.get_line(data_file,status,length);
                    for i in length+1..24 loop
```

```
                    status(i):=' ';
                  end loop;
                  text_io.get_line(data_file,designer,length);
                  for i in length+1..24 loop
                    designer(i):=' ';
                  end loop;
                  text_io.get_line(data_file,deadline,length);
                  for i in length+1..24 loop
                    deadline(i):=' ';
                  end loop;
                text_io.get_line(data_file,start_time,length);
                  for i in length+1..24 loop
                    start_time(i):=' ';
                  end loop;
               text_io.get_line(data_file,finish_time,length);
                  for i in length+1..24 loop
                    finish_time(i):=' ';
                  end loop;
             text_io.get_line(data_file,primary_input,length);
                  for i in length+1..64 loop
                    primary_input(i):=' ';
                  end loop;

-- read number of secondary inputs followed by secondary inputs

                 get(data_file,counter); skip_line(data_file);
                  if counter > 0 then
                  for i in 1..counter loop

        text_io.get_line(data_file,secondary_inputs(i).all,leng
        th);
                    for j in length+1..64 loop
                      secondary_inputs(i)(j):=' ';
                    end loop;

        TAE_Wpt.Wpt_SetStringConstraints(show_info.panel_id,
                           "sec_input", TaeInt(i),
        secondary_inputs);

                  end loop;
                  end if;

-- now read number of affected modules followed by affectecd
        modules

                 get(data_file,counter); skip_line(data_file);
```

```
                    if counter > 0 then
                    for i in 1..counter loop

        text_io.get_line(data_file,affected_modules(i).all,leng
        th);
                        for j in length+1..64 loop
                          affected_modules(i)(j):=' ';
                        end loop;

        TAE_Wpt.Wpt_SetStringConstraints(show_info.panel_id,
                       "affected", TaeInt(i), affected_modules);
                    end loop;
                    end if;

                    get(data_file,counter);
                    if counter > 0 then skip_line(data_file);

        get_line(data_file,sub_steps,length);
                                  else skip_line(data_file);
                       end if;

                    get(data_file,counter);
                    if counter > 0 then skip_line(data_file);

        get_line(data_file,predecessors,length);
                       end if;

text_io.put_line("done reading, writing to panel");

text_io.put_line("done writing to panel, closing file");

                    text_io.CLOSE(data_file);

-- now write all the new step information into the TAE window

        Tae_Wpt.Wpt_SetString(show_info.panel_id,"base_version",
                                          base_version);
        Tae_Wpt.Wpt_SetIntg(show_info.panel_id,"est_duration",
                                          Taeint(est_duration));
        Tae_Wpt.Wpt_SetString(show_info.panel_id,"exp_level",
                                          expertise_level);
        Tae_Wpt.Wpt_SetIntg(show_info.panel_id,"priority",
                                          Taeint(priority));
        Tae_Wpt.Wpt_SetString(show_info.panel_id,"designer",
                                          designer);
        Tae_Wpt.Wpt_SetString(show_info.panel_id,"deadline",
```

```
                                                deadline);
        Tae_Wpt.Wpt_SetString(show_info.panel_id,"start_time",
                                                start_time);
        Tae_Wpt.Wpt_SetString(show_info.panel_id,"finish_time",
                                                finish_time);
        Tae_Wpt.Wpt_SetString(show_info.panel_id,"pri_input",
                                                primary_input);
        Tae_Wpt.Wpt_SetIntg(show_info.panel_id,"step_number",
                                                value(1));
        Tae_Wpt.Wpt_SetString(show_info.panel_id,"sub_steps",
                                                sub_steps);
        Tae_Wpt.Wpt_SetString(show_info.panel_id,"predecessors",
                                                predecessors);

        exception

           when text_io.NAME_ERROR =>
              text_io.put_line("ERROR: non-existent transfer file
           (probably 'ddbdisplay').");
           when text_io.END_ERROR =>
                text_io.put_line("ERROR: corrupt transfer file
           (probably 'ddbdisplay').");
              if is_open(data_file) then close(data_file); end if;
           when text_io.STATUS_ERROR =>
              if is_open(data_file) then close(data_file); end if;


    end showstep_s_select_item;

procedure show_step_number (info : in tae_wpt.event_context_ptr)
          is

        begin

           null;

    end show_step_number;

procedure show_show_finish (info : in tae_wpt.event_context_ptr)
          is
    value : array (1..1) of string (1..tae_taeconf.STRINGSIZE);
    count : taeint;

        begin
        text_io.put ("Panel show, parm show_finish: value = ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
```

```
        if count <= 0 then
           text_io.put_line ("none");
        else
           tae_vm.Vm_Extract_SVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (value(1));
        end if;
        tae_wpt.Wpt_PanelErase(info.panel_id);
    end show_show_finish;

procedure text1_display_done (info : in tae_wpt.event_context_ptr)
          is

        begin

        SELECTOR := 13;  -- this selection erases the transfer file
                         -- (currently 'ddbdisplay') for its next use
          ECS1;

          tae_wpt.Wpt_PanelErase(info.panel_id);

    end text1_display_done;

procedure text1_text_item_1 (info : in tae_wpt.event_context_ptr)
          is

        begin

          null;

    end text1_text_item_1;

procedure steptype_type_selection (info : in
          tae_wpt.event_context_ptr) is
    value : array (1..1) of string (1..tae_taeconf.STRINGSIZE);
    count : taeint;

        begin
SELECTOR := 2;

       text_io.put ("Panel steptype, parm type_selection: value =
          ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
        else
           tae_vm.Vm_Extract_SVAL (info.parm_ptr, 1, value(1));
```

```
           text_io.put_line (value(1));
        end if;
        if (FALSE) then null;
        elsif s_equal (value(1), "all") then
Name3(1..3) := "all";
                    for i in 4..64 loop
                        Name3(i):=' ';
                    end loop;
ECS1;
           if info.panel_id = NULL_PANEL_ID then
               tae_wpt.Wpt_NewPanel ("", info.target, info.view,
             X_Windows.Null_Window, info, tae_wpt.WPT_INVISIBLE,
                   info.panel_id);
           else
               tae_wpt.Wpt_SetPanelState (
                  info.panel_id, tae_wpt.WPT_INVISIBLE);
           end if;
           if text1_info.panel_id = NULL_PANEL_ID then
               tae_wpt.Wpt_NewPanel ("", text1_info.target,
        text1_info.view,
                   X_Windows.Null_Window, text1_info,
        tae_wpt.WPT_PREFERRED,
                   text1_info.panel_id);
           else
               tae_wpt.Wpt_SetPanelState (
                  text1_info.panel_id, tae_wpt.WPT_PREFERRED);
           end if;
        elsif s_equal (value(1), "top") then
Name3(1..3) := "top";
                    for i in 4..64 loop
                        Name3(i):=' ';
                    end loop;
ECS1;
           if info.panel_id = NULL_PANEL_ID then
               tae_wpt.Wpt_NewPanel ("", info.target, info.view,
             X_Windows.Null_Window, info, tae_wpt.WPT_INVISIBLE,
                   info.panel_id);
           else
               tae_wpt.Wpt_SetPanelState (
                  info.panel_id, tae_wpt.WPT_INVISIBLE);
           end if;
           if text1_info.panel_id = NULL_PANEL_ID then
               tae_wpt.Wpt_NewPanel ("", text1_info.target,
        text1_info.view,
                   X_Windows.Null_Window, text1_info,
        tae_wpt.WPT_PREFERRED,
```

```
                    text1_info.panel_id);
            else
                tae_wpt.Wpt_SetPanelState (
                    text1_info.panel_id, tae_wpt.WPT_PREFERRED);
            end if;
        elsif s_equal (value(1), "proposed") then
Name3(1..8) := "proposed";
                    for i in 9..64 loop
                        Name3(i):=' ';
                    end loop;
ECS1;
            if info.panel_id = NULL_PANEL_ID then
                tae_wpt.Wpt_NewPanel ("", info.target, info.view,
              X_Windows.Null_Window, info, tae_wpt.WPT_INVISIBLE,
                    info.panel_id);
            else
                tae_wpt.Wpt_SetPanelState (
                    info.panel_id, tae_wpt.WPT_INVISIBLE);
            end if;
            if text1_info.panel_id = NULL_PANEL_ID then
                tae_wpt.Wpt_NewPanel ("", text1_info.target,
          text1_info.view,
                    X_Windows.Null_Window, text1_info,
          tae_wpt.WPT_PREFERRED,
                    text1_info.panel_id);
            else
                tae_wpt.Wpt_SetPanelState (
                    text1_info.panel_id, tae_wpt.WPT_PREFERRED);
            end if;
        elsif s_equal (value(1), "approved") then
Name3(1..8) := "approved";
                    for i in 9..64 loop
                        Name3(i):=' ';
                    end loop;
ECS1;
            if info.panel_id = NULL_PANEL_ID then
                tae_wpt.Wpt_NewPanel ("", info.target, info.view,
              X_Windows.Null_Window, info, tae_wpt.WPT_INVISIBLE,
                    info.panel_id);
            else
                tae_wpt.Wpt_SetPanelState (
                    info.panel_id, tae_wpt.WPT_INVISIBLE);
            end if;
            if text1_info.panel_id = NULL_PANEL_ID then
                tae_wpt.Wpt_NewPanel ("", text1_info.target,
          text1_info.view,
```

```
                X_Windows.Null_Window, text1_info,
        tae_wpt.WPT_PREFERRED,
                text1_info.panel_id);
          else
              tae_wpt.Wpt_SetPanelState (
                 text1_info.panel_id, tae_wpt.WPT_PREFERRED);
          end if;
      elsif s_equal (value(1), "scheduled") then
Name3(1..9) := "scheduled";
                 for i in 10..64 loop
                     Name3(i):=' ';
                 end loop;
ECS1;
          if info.panel_id = NULL_PANEL_ID then
              tae_wpt.Wpt_NewPanel ("", info.target, info.view,
            X_Windows.Null_Window, info, tae_wpt.WPT_INVISIBLE,
                info.panel_id);
          else
              tae_wpt.Wpt_SetPanelState (
                 info.panel_id, tae_wpt.WPT_INVISIBLE);
          end if;
          if text1_info.panel_id = NULL_PANEL_ID then
              tae_wpt.Wpt_NewPanel ("", text1_info.target,
        text1_info.view,
                X_Windows.Null_Window, text1_info,
        tae_wpt.WPT_PREFERRED,
                text1_info.panel_id);
          else
              tae_wpt.Wpt_SetPanelState (
                 text1_info.panel_id, tae_wpt.WPT_PREFERRED);
          end if;
      elsif s_equal (value(1), "assigned") then
Name3(1..8) := "assigned";
                 for i in 9..64 loop
                     Name3(i):=' ';
                 end loop;
ECS1;
          if info.panel_id = NULL_PANEL_ID then
              tae_wpt.Wpt_NewPanel ("", info.target, info.view,
            X_Windows.Null_Window, info, tae_wpt.WPT_INVISIBLE,
                info.panel_id);
          else
              tae_wpt.Wpt_SetPanelState (
                 info.panel_id, tae_wpt.WPT_INVISIBLE);
          end if;
          if text1_info.panel_id = NULL_PANEL_ID then
```

```
                tae_wpt.Wpt_NewPanel ("", text1_info.target,
          text1_info.view,
                   X_Windows.Null_Window, text1_info,
          tae_wpt.WPT_PREFERRED,
                   text1_info.panel_id);
             else
                tae_wpt.Wpt_SetPanelState (
                   text1_info.panel_id, tae_wpt.WPT_PREFERRED);
             end if;
          elsif s_equal (value(1), "completed") then
Name3(1..9) := "completed";
                    for i in 10..64 loop
                        Name3(i):=' ';
                    end loop;
ECS1;
             if info.panel_id = NULL_PANEL_ID then
                tae_wpt.Wpt_NewPanel ("", info.target, info.view,
               X_Windows.Null_Window, info, tae_wpt.WPT_INVISIBLE,
                   info.panel_id);
             else
                tae_wpt.Wpt_SetPanelState (
                   info.panel_id, tae_wpt.WPT_INVISIBLE);
             end if;
             if text1_info.panel_id = NULL_PANEL_ID then
                tae_wpt.Wpt_NewPanel ("", text1_info.target,
          text1_info.view,
                   X_Windows.Null_Window, text1_info,
          tae_wpt.WPT_PREFERRED,
                   text1_info.panel_id);
             else
                tae_wpt.Wpt_SetPanelState (
                   text1_info.panel_id, tae_wpt.WPT_PREFERRED);
             end if;
          elsif s_equal (value(1), "abandoned") then
Name3(1..9) := "abandoned";
                    for i in 10..64 loop
                        Name3(i):=' ';
                    end loop;
ECS1;
             if info.panel_id = NULL_PANEL_ID then
                tae_wpt.Wpt_NewPanel ("", info.target, info.view,
               X_Windows.Null_Window, info, tae_wpt.WPT_INVISIBLE,
                   info.panel_id);
             else
                tae_wpt.Wpt_SetPanelState (
                   info.panel_id, tae_wpt.WPT_INVISIBLE);
```

```
            end if;
            if text1_info.panel_id = NULL_PANEL_ID then
                tae_wpt.Wpt_NewPanel ("", text1_info.target,
          text1_info.view,
                    X_Windows.Null_Window, text1_info,
          tae_wpt.WPT_PREFERRED,
                    text1_info.panel_id);
            else
                tae_wpt.Wpt_SetPanelState (
                    text1_info.panel_id, tae_wpt.WPT_PREFERRED);
            end if;
        end if;
    end steptype_type_selection;

procedure editnum_s_select_item (info : in
          tae_wpt.event_context_ptr) is
    value : array (1..1) of taeint;
    count : taeint;

        begin
       text_io.put ("Panel editnum, parm s_select_item: value = ");
        tae_vm.Vm_Extract_Count (info.parm_ptr, count);
        if count <= 0 then
           text_io.put_line ("none");
        else
           tae_vm.Vm_Extract_IVAL (info.parm_ptr, 1, value(1));
           text_io.put_line (taeint'image(value(1)));
        end if;
        tae_wpt.Wpt_PanelErase(info.panel_id);

-- write step info to transfer file

taeint_io.put(Name3,(value(1)));
step_number := integer(value(1));
SELECTOR := 3;
ECS1;
editing_or_creating := editing;
CLEAR_STEP_INFO;

-- create new panel

        if editstep_info.panel_id = NULL_PANEL_ID then
            tae_wpt.Wpt_NewPanel ("", editstep_info.target,
          editstep_info.view,
               X_Windows.Null_Window, editstep_info,
          tae_wpt.WPT_PREFERRED,
```

```
            editstep_info.panel_id);
        else
            tae_wpt.Wpt_SetPanelState (
               editstep_info.panel_id, tae_wpt.WPT_PREFERRED);
        end if;

-- make sure that the panel items are cleared

                    for i in 1..MAX_SEC_INPUTS loop

          TAE_Wpt.Wpt_SetStringConstraints(editstep_info.panel_id
          ,
                                "sec_input", TaeInt(i),
          secondary_inputs);


          TAE_Wpt.Wpt_SetStringConstraints(editstep_info.panel_id
          ,
                        "affected", TaeInt(i), affected_modules);

                    end loop;


          Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"base_vers
          ion",
                                              base_version);
       Tae_Wpt.Wpt_SetIntg(editstep_info.panel_id,"est_duration",
                                              Taeint(est_duration));
        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"exp_level",
                                              expertise_level);
        Tae_Wpt.Wpt_SetIntg(editstep_info.panel_id,"priority",
                                              Taeint(priority));
        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"designer",
                                              designer);
        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"deadline",
                                              deadline);
       Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"start_time",
                                              start_time);
      Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"finish_time",
                                              finish_time);
        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"pri_input",
                                              primary_input);
        Tae_Wpt.Wpt_SetIntg(editstep_info.panel_id,"step_number",
                                              taeint(step_number));
        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"sub_steps",
                                              sub_steps);
```

```
        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"predecess
        ors",
                                          predecessors);


-- now read the file

text_io.put_line("opening file");

                    text_io.OPEN(data_file,
          text_io.IN_FILE,"ddbdisplay");

text_io.put_line("reading from file");

                text_io.get_line(data_file,base_version,length);
                    for i in length+1..64 loop
                       base_version(i):=' ';
                    end loop;
                    get(data_file,est_duration);
                    skip_line(data_file);
                    get(data_file,priority);
                    skip_line(data_file);

          text_io.get_line(data_file,expertise_level,length);
                    for i in length+1..24 loop
                       expertise_level(i):=' ';
                    end loop;
                    text_io.get_line(data_file,status,length);
                    for i in length+1..24 loop
                       status(i):=' ';
                    end loop;
                    text_io.get_line(data_file,designer,length);
                    for i in length+1..24 loop
                       designer(i):=' ';
                    end loop;
                    text_io.get_line(data_file,deadline,length);
                    for i in length+1..24 loop
                       deadline(i):=' ';
                    end loop;
                  text_io.get_line(data_file,start_time,length);
                    for i in length+1..24 loop
                       start_time(i):=' ';
                    end loop;
                 text_io.get_line(data_file,finish_time,length);
                    for i in length+1..24 loop
```

```
                    finish_time(i):=' ';
                 end loop;
              text_io.get_line(data_file,primary_input,length);
                 for i in length+1..64 loop
                    primary_input(i):=' ';
                 end loop;

-- read number of secondary inputs followed by secondary inputs

                get(data_file,counter); skip_line(data_file);
                 if counter > 0 then
                 for i in 1..counter loop

        text_io.get_line(data_file,secondary_inputs(i).all,leng
        th);
                     for j in length+1..64 loop
                       secondary_inputs(i)(j):=' ';
                     end loop;

        TAE_Wpt.Wpt_SetStringConstraints(editstep_info.panel_id
        ,
                          "sec_input", TaeInt(i),
        secondary_inputs);

                 end loop;
                 end if;

-- now read number of affected modules followed by affectecd
        modules

                get(data_file,counter); skip_line(data_file);
                 if counter > 0 then
                 for i in 1..counter loop

        text_io.get_line(data_file,affected_modules(i).all,leng
        th);
                     for j in length+1..64 loop
                       affected_modules(i)(j):=' ';
                     end loop;

        TAE_Wpt.Wpt_SetStringConstraints(editstep_info.panel_id
        ,
                    "affected", TaeInt(i), affected_modules);
                 end loop;
                 end if;
```

```
                    get(data_file,counter);
                    if counter > 0 then skip_line(data_file);

          get_line(data_file,sub_steps,length);
                                  else skip_line(data_file);
                       end if;

                    get(data_file,counter);
                    if counter > 0 then skip_line(data_file);

          get_line(data_file,predecessors,length);
                       end if;

text_io.put_line("done reading, writing to panel");

text_io.put_line("done writing to panel, closing file");

                    text_io.CLOSE(data_file);

-- now write all the step information into the TAE window


          Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"base_vers
          ion",
                                              base_version);
       Tae_Wpt.Wpt_SetIntg(editstep_info.panel_id,"est_duration",
                                              Taeint(est_duration));
        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"exp_level",
                                              expertise_level);
        Tae_Wpt.Wpt_SetIntg(editstep_info.panel_id,"priority",
                                              Taeint(priority));
        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"designer",
                                              designer);
        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"deadline",
                                              deadline);
       Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"start_time",
                                              start_time);
      Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"finish_time",
                                              finish_time);
        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"pri_input",
                                              primary_input);
        Tae_Wpt.Wpt_SetIntg(editstep_info.panel_id,"step_number",
                                              value(1));
        Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"sub_steps",
                                              sub_steps);
```

```
          Tae_Wpt.Wpt_SetString(editstep_info.panel_id,"predecess
          ors",
                                              predecessors);

        exception

            when text_io.NAME_ERROR =>
               text_io.put_line("ERROR: non-existent transfer file
            (probably 'ddbdisplay').");
            when text_io.END_ERROR =>
                 text_io.put_line("ERROR: corrupt transfer file
            (probably 'ddbdisplay').");
               if is_open(data_file) then close(data_file); end if;
            when text_io.STATUS_ERROR =>
               if is_open(data_file) then close(data_file); end if;


    end editnum_s_select_item;

--        END EVENT_HANDLERs


end ddb_interface_support;

-- ENDFILE: ddb_interface_support_body.a

-------------------------------------------------------------------
          ---------


--
-- Main Program
--
  begin

    f_force_lower (FALSE);    -- permit upper/lowercase file names
    tae_wpt.Wpt_Init ("",theDisplay);
    tae_wpt.Wpt_NewEvent (wptEvent);

    initializePanels ("ddb_interface.res");   -- single call

--      main event loop

EVENT_LOOP:
    loop
        tae_wpt.Wpt_NextEvent (wptEvent, etype);     -- get next
          event
```

```
-- NOTE: This case statement includes STUBs for non-WPT_PARM_EVENT
         events.

       case etype is

          when wpt_eventtype'first .. -1 => null;
              -- iterate loop on Wpt_NextEvent error

-- TYPICAL CASE: Panel Event (WPT_PARM_EVENT)

          when tae_wpt.WPT_PARM_EVENT =>
          -- You can comment out the following "put" call.
          -- The appropriate EVENT_HANDLER finishes the message.
                  text_io.put ( "Event: WPT_PARM_EVENT, " );

          --      Panel event has occurred.
          --      Get parm name and then call appropriate
        EVENT_HANDLER.
          --
          --                              CAUTION:
          --      DO NOT call Wpt_Extract_Parm_xEvent from any
        other branch
          --      of this "case" statement or you'll get
        "storage_error".
          --
               tae_wpt.Wpt_Extract_Context (wptEvent, user_ptr);
                  tae_wpt.Wpt_Extract_Parm (wptEvent,
        user_ptr.parm_name);
                  tae_wpt.Wpt_Extract_Data (wptEvent,
        user_ptr.datavm_ptr);
                  tae_vm.Vm_Find (user_ptr.datavm_ptr,
        user_ptr.parm_name,
                                  user_ptr.parm_ptr);

          -- dummy if to ease code generation
              if (FALSE) then null;

                  -- WPT_PARM_EVENT, BEGIN panel main

                  elsif tae_wpt."=" (user_ptr, main_info)  then
                  if (FALSE) then null;    -- another dummy if
                    -- determine appropriate EVENT_HANDLER for
        this item
                     elsif s_equal ("selection_1",
        user_ptr.parm_name) then
```

```
                    main_selection_1 (user_ptr);
            end if;     -- END panel main

            -- WPT_PARM_EVENT, BEGIN panel editstep

        elsif tae_wpt."=" (user_ptr, editstep_info)  then
            if (FALSE) then null;     -- another dummy if
              -- determine appropriate EVENT_HANDLER for
this item
                elsif s_equal ("base_version",
user_ptr.parm_name) then
                    editstep_base_version (user_ptr);
          elsif s_equal ("pri_input", user_ptr.parm_name)
then
                    editstep_pri_input (user_ptr);
                elsif s_equal ("predecessors",
user_ptr.parm_name) then
                    editstep_predecessors (user_ptr);
           elsif s_equal ("priority", user_ptr.parm_name)
then
                    editstep_priority (user_ptr);
          elsif s_equal ("exp_level", user_ptr.parm_name)
then
                    editstep_exp_level (user_ptr);
           elsif s_equal ("deadline", user_ptr.parm_name)
then
                    editstep_deadline (user_ptr);
                elsif s_equal ("est_duration",
user_ptr.parm_name) then
                    editstep_est_duration (user_ptr);
          elsif s_equal ("sec_input", user_ptr.parm_name)
then
                    editstep_sec_input (user_ptr);
           elsif s_equal ("affected", user_ptr.parm_name)
then
                    editstep_affected (user_ptr);
             elsif s_equal ("return", user_ptr.parm_name)
then
                    editstep_return (user_ptr);
                elsif s_equal ("apply_step",
user_ptr.parm_name) then
                    editstep_apply_step (user_ptr);
                elsif s_equal ("cancel_step",
user_ptr.parm_name) then
                    editstep_cancel_step (user_ptr);
            end if;     -- END panel editstep
```

```
            -- WPT_PARM_EVENT, BEGIN panel editteam

         elsif tae_wpt."=" (user_ptr, editteam_info)  then
            if (FALSE) then null;     -- another dummy if
              -- determine appropriate EVENT_HANDLER for
this item
           elsif s_equal ("name", user_ptr.parm_name) then
                    editteam_name (user_ptr);
            elsif s_equal ("ex_opt", user_ptr.parm_name)
then
                    editteam_ex_opt (user_ptr);
           elsif s_equal ("d_cancel", user_ptr.parm_name)
then
                    editteam_d_cancel (user_ptr);
          elsif s_equal ("designers", user_ptr.parm_name)
then
                    editteam_designers (user_ptr);
                elsif s_equal ("selection_3",
user_ptr.parm_name) then
                    editteam_selection_3 (user_ptr);
           end if;     -- END panel editteam

           -- WPT_PARM_EVENT, BEGIN panel confirm

         elsif tae_wpt."=" (user_ptr, confirm_info)  then
            if (FALSE) then null;     -- another dummy if
              -- determine appropriate EVENT_HANDLER for
this item
           elsif s_equal ("yes", user_ptr.parm_name) then
                    confirm_yes (user_ptr);
            elsif s_equal ("no", user_ptr.parm_name) then
                    confirm_no (user_ptr);
           end if;     -- END panel confirm

           -- WPT_PARM_EVENT, BEGIN panel s_select

         elsif tae_wpt."=" (user_ptr, s_select_info)  then
            if (FALSE) then null;     -- another dummy if
              -- determine appropriate EVENT_HANDLER for
this item
                elsif s_equal ("s_select_item",
user_ptr.parm_name) then
                    s_select_s_select_item (user_ptr);
           end if;     -- END panel s_select
```

```
            -- WPT_PARM_EVENT, BEGIN panel showstep

          elsif tae_wpt."=" (user_ptr, showstep_info)  then
             if (FALSE) then null;     -- another dummy if
               -- determine appropriate EVENT_HANDLER for
this item
                elsif s_equal ("s_select_item",
user_ptr.parm_name) then
                    showstep_s_select_item (user_ptr);
             end if;     -- END panel showstep

             -- WPT_PARM_EVENT, BEGIN panel show

             elsif tae_wpt."=" (user_ptr, show_info)  then
             if (FALSE) then null;     -- another dummy if
               -- determine appropriate EVENT_HANDLER for
this item
                elsif s_equal ("step_number",
user_ptr.parm_name) then
                    show_step_number (user_ptr);
                elsif s_equal ("show_finish",
user_ptr.parm_name) then
                    show_show_finish (user_ptr);
             end if;     -- END panel show

             -- WPT_PARM_EVENT, BEGIN panel text1

           elsif tae_wpt."=" (user_ptr, text1_info)  then
             if (FALSE) then null;     -- another dummy if
               -- determine appropriate EVENT_HANDLER for
this item
                elsif s_equal ("display_done",
user_ptr.parm_name) then
                    text1_display_done (user_ptr);
                elsif s_equal ("text_item_1",
user_ptr.parm_name) then
                    text1_text_item_1 (user_ptr);
             end if;     -- END panel text1

             -- WPT_PARM_EVENT, BEGIN panel steptype

          elsif tae_wpt."=" (user_ptr, steptype_info)  then
             if (FALSE) then null;     -- another dummy if
               -- determine appropriate EVENT_HANDLER for
this item
                elsif s_equal ("type_selection",
```

```
        user_ptr.parm_name) then
                        steptype_type_selection (user_ptr);
                end if;     -- END panel steptype

                -- WPT_PARM_EVENT, BEGIN panel editnum

              elsif tae_wpt."=" (user_ptr, editnum_info)  then
                if (FALSE) then null;     -- another dummy if
                  -- determine appropriate EVENT_HANDLER for
        this item
                   elsif s_equal ("s_select_item",
        user_ptr.parm_name) then
                        editnum_s_select_item (user_ptr);
                end if;     -- END panel editnum

                else
              text_io.put_line ("unexpected event from wpt!");
                 exit; -- or raise an exception, but compiler
        warns if no exit
              end if;

          when tae_wpt.WPT_FILE_EVENT =>
                    text_io.put_line ("STUB: Event
        WPT_FILE_EVENT");

                  -- Use Wpt_AddEvent and Wpt_RemoveEvent and
                   -- Wpt_Extract_EventSource and
        Wpt_Extract_EventMask

          when tae_wpt.WPT_TIMEOUT_EVENT =>
                    text_io.put_line ("STUB: Event
        WPT_TIMEOUT_EVENT");

                   -- Use Wpt_SetTimeOut for this

-- LEAST LIKELY cases follow:

          when tae_wpt.WPT_WINDOW_EVENT => null ;

                  -- WPT_WINDOW_EVENT can be caused by user
        acknowledgement
                -- of a Wpt_PanelMessage or windows which you
                  -- directly create with X (not TAE panels).
              -- You MIGHT want to use Wpt_Extract_xEvent_Type
        here.
                   --
```

```
                    -- DO NOT use Wpt_Extract_Parm_xEvent since
        this is not
                     -- a WPT_PARM_EVENT; you'll get a "storage
        error".

          when tae_wpt.WPT_HELP_EVENT =>          -- OR null ;
                       text_io.put("ERROR: WPT_HELP_EVENT: ");
                     text_io.put_line("should never see; reserved
        for TAE use");

          when tae_wpt.WPT_INTERRUPT_EVENT =>     -- OR null ;
                      text_io.put("ERROR: WPT_INTERRUPT_EVENT: ");
                     text_io.put_line("should never see; reserved
        for TAE use");

          when OTHERS =>
                text_io.put ("FATAL ERROR: Unknown Wpt_NextEvent
        Event Type: ");
                     text_io.put (wpt_eventtype'image(etype) ) ;
                       text_io.put_line (" ... Forcing exit.");
                       exit;  -- or raise an exception


      end case;     -- NOTE: Do not add statements between here
        and "end loop EVENT_LOOP"

  end loop EVENT_LOOP;

end ddb_interface;
```

D. TEST DATA AND TEST RESULTS

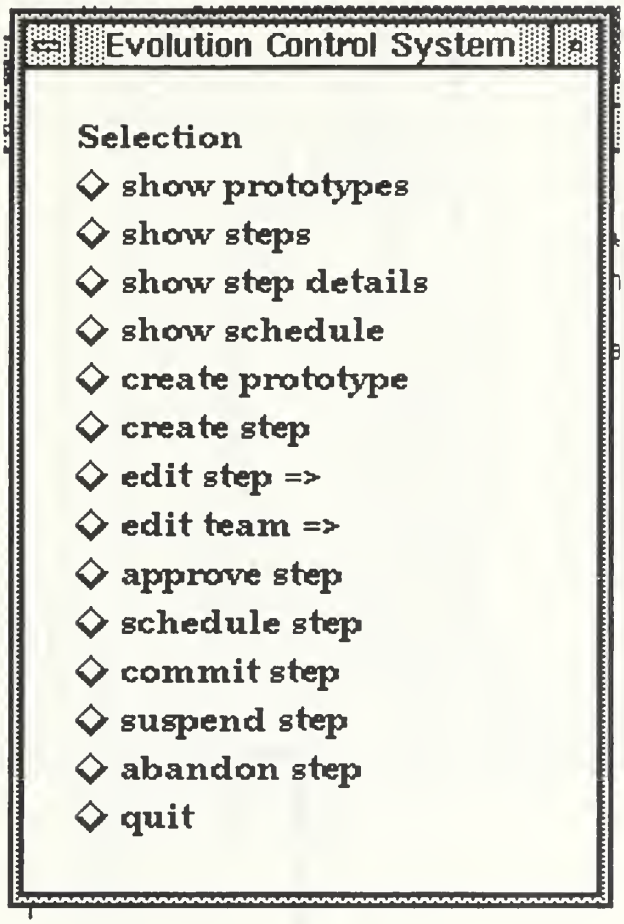


FIGURE 33. Screen image of the ECS main menu (manager)

FIGURE 34. Screen image of the initial designer pool data

FIGURE 35. Screen image of step 1 details after it has been created

FIGURE 36. Screen image of step 2 details after it has been created

FIGURE 37. Screen image of step 3 details after it has been created

FIGURE 38. Screen image of step 4 details after it has been created

FIGURE 39. Screen image of step 5 details after it has been created

FIGURE 40. Screen image of step 1 details after it has been approved

FIGURE 41. Screen image of step 6 details (substep of step 1)

FIGURE 42. Screen image of step 7 details (substep of step 1)

FIGURE 43. Screen image of step 8 details (substep of step 1)

FIGURE 44. Image of step 2 details after it has been approved)

FIGURE 45. Image of step 9 details (substep of step 2)

FIGURE 46. Image of step 10 details (substep of step 2)

FIGURE 47. Image of step 4 details after it has been approved

FIGURE 48. Image of step 11 details (substep of step 4)

FIGURE 49. Image of step 12 details (substep of step 4)

FIGURE 50. Image of step 13 details (substep of step 4)

FIGURE 51. Screen image of step 6 after its completion

FIGURE 52. Screen image of step 8 after its completion

FIGURE 53. Screen image of step 7 after its completion

FIGURE 54. Screen image of step 1 after its completion

FIGURE 55. Screen image of step 2 after the completion of step 1

FIGURE 56. Screen image of step 3 after it has been scheduled

FIGURE 57. Screen image of step 15 after it has been scheduled

FIGURE 58. Screen image of step 3 after the completion of step 1

VIII. LIST OF REFERENCES

[1] "Aide~de~Camp Software Management System Technical Reference Guide", Software Maintenance & Development Systems INC. Version 7.21, 1991.

[2] Andreas Drexl, "Scheduling of Project Networks by Job Assignment", Management Science, Vol. 37. No. 12, Dec. 1991, pp. 1590-1602.

[3] Arthur L. J., "Software Evolution - The Software Maintenance Challenge", John Wiley and Sons, 1988.

[4] Badrinath B. R. and Ramamrithan K., "Semantics-Based Concurrency Control: Beyond Commutativity", ACM Trans. on Database Systems Vol. 17, NO 1, March 1992.

[5] Badr Salah and Luqi, "A Version and Configuration Model for Software Evolution", Proceedings of the Fifth International Conference on Software Engineering and Knowledge Engineering (SEKE'93), San Francisco Bay, June 16-18, 1993. pp.225-227

[6] Badr S. and Berzins V., "A Design Management and Job Assignment System", Technical Report, Computer Science Department, Naval Postgraduate School, NPS-CS-020-1992

[7] Baker K. R., Su Zaw-Sing, "Sequencing with Due-Dates and Early Start Times to Minimize Maximum Tardiness", Naval Research Logistic Quarterly, Vol. 21, NO. 1, March 1974. pp. 171-176.

[8] Balas E. and Saltzman M. "An Algorithm For The Three-Index Assignment Problem", Operations Research, vol. 39, No. 1, Jan.-Feb. 1991.

[9] Barghouti N. S. and Kaiser G. E., "Concurrency Control in Advanced Database Applications", ACM Computing Surveys, Vol. 23, No. 3, September 1991. pp. 269-317.

[10] Bean J., Birge J., Mittenthal J., and Noon C., "Matchup Scheduling with Multiple Resources release Dates and Disruptions", Operations Research, vol. 39, No. 3, May-June 1991.

[11] Berzins and Luqi, "Software Engineering with Abstractions", Addison-Wesley 1990

[12] Bezalel Gavish and Hasan Pirkul, "Algorithms for The Multi-Resource Generalized Assignment Problem", Management Science, Vol. 37. No. 6, Dec. 1991, pp. 695-713.

[13] Biyabani S. R., Stankovic J. A., Ramamritham k., "The Integration of Deadline and Criticalness in Hard Real-Time Scheduling", Real-Time Systems Workshop, May 1988.

[14] Borison E., "A Model of Software Manufacture", Proceedings of the international workshop, Trondheim, Norway, June 1986, pp. 197-220.

[15] Bratley P., Florian M., Robillard P., "Scheduling with Earliest Start and Due Date Constraints", Naval Research Logistic Quarterly, Vol. 18, NO. 4, December 1971. pp. 511-519.

[16] Campbell R. H., Terwilliger R. B. "The SAGA Approach to Automated Project Management", in Advanced Programming Environment, Springer-Verlag, 1986, pp. 142-155.

[17] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli, "Fundamentals of Software Engineering", Printice Hall, 1991.

[18] Cheng S., Stankovic J. A., Ramamritham K., "Scheduling Algorithms for Hard Real-Time Systems - A Brief Survey", COINS Technical Report 87-55, Dept. of Computer and Information Science, University of Massachusetts, June 1987.

[19] Coffman E. G., "Computer and Job-Shop Scheduling Theory", John Wiley & Sons 1976.

[20] Cohen E. S., Soni D. A., Gluecker R., Hasling W. M., Schwanke R. W., and Wagner M. E., "Version Management in Gypsy", Proceedings of the ACM SIGSOFT/SIGPLAN, Nov. 28-30, 1988. pp. 201-215.

[21] Colin J. and Chretienne P., "C. P. M. Scheduling with Small Communication Delays and Task Duplication", Operations Research, vol. 39, No. 4, July-Aug. 1991.

[22] Dampier D., "A Model for Merging Different Versions of a PSDL Program", MS thesis, Computer Science Department, Naval Postgraduate School, June 1990.

[23] Dampier D., Luqi, "A Model for Merging Software Prototypes", Technical Report, NPS CS-92-014.

[24] Falkenberg B., "Configuration Management in a Large (SW) Development", Proceedings of 2nd International Workshop on Software Configuration Management, Princeton, New Jersey, Oct. 24,1989. pp. 34-37.

[25] Fan B. H., "Evaluations of Some Scheduling Algorithms for Hard Real-Time Systems" MS thesis, Computer Science Department, Naval Postgraduate School, June 1990.

[26] Feiler P. H., "Configuration Management Models in Commercial Environments", Technical Report CMU-91-TR-7, ESD-91-TR-7, 1991.

[27] Feldman S. I., "Software Configuration Management: Past Uses and Future Challenges" Proceedings of 3rd European Software Engineering Conference, ESEC '91, Milan, Italy, October 1991

[28] Fischetti M., Martello S. and Toth P. "The Fixed Job Scheduling Problem with Spread Time Constraints", Operations Research, vol. 35, No. 6, Nov.-Dec. 1987.

[29] Grady R. B., "Measuring and Managing Software Maintenance", IEEE Software, Sept. 1987, pp. 35-45.

[30] Gustafson D. A., et al, "Software Maintenance Models", Software Maintenance and Computers, IEEE Computer Society Press Tutorial, 1990. pp.23-35.

[31] Gustavsson A., "Maintaining the Evolution of Software Objects in an Integrated Environment", Proceedings of 2nd International Workshop on Software Configuration Management, Princeton, New Jersey, Oct. 24,1989. pp. 114-117.

[32] Heimbigner D. and Krane S., "A Graph Transform Model for Configuration Management Environments", Proceedings of the ACM SIGSOFT/ SIGPLAN, Nov. 28-30, 1988. pp. 216-225.

[33] Hillier & Lieberman., "Introduction to Operation Research", fourth Edition, 1986.

[34] Hong K. and Leung J., "On-Line Scheduling of Real-Time Tasks" Real-Time Systems Workshop, May 1988.

[35] Horn W. A., "Some Simple Scheduling Algorithms" Naval Research Logistic Quarterly, Vol. 21, NO. 1, March 1974. pp. 177-185.

[36] "IEEE Guide to Software Configuration Management", Std 1042-1987, American National Standards Institute/IEEE, New York, 1988.

[37] Kaiser G. E., and Perry D. E., "Workspaces and Experimental Databases: Automated Support for Software Maintenance and Evolution", Proceedings of IEEE Conference on Software Maintenance 1987. pp. 108-114.

[38] Kaiser G. E., and Perry D. E., and Schell W. M., "Infuse: Fusing Integration Test Management with Change Management", Proceedings of the Thirteenth Annual International Computer Software & Applications Conference, Orlando, FL, September 20-22, 1989.

[39] Kate R. H., "Toward a Unified Framework for Version Modeling in Engineering Databases", ACM Computing Surveys, VOL. 22, NO. 4, December 1990.

[40] Ketabchi M. and Berzins V., "Generalization per Category: Theory and Application", Proceedings Int. Conf. on Information Systems, 1986.

[41] Ketabchi M. A., "On the Management of Computer Aided Design Database", Ph. D. Dissertation, University of Minnesota, Nov. 1985.

[42] Lacroix M. and Lavency P., "The Change Request Process", Proceedings of 2nd International Workshop on Software Configuration Management, Princeton, New Jersey, Oct. 24,1989. pp. 122-125.

[43] Leung Joseph Y-T., Young G. H., "Preemptive Scheduling to Minimize Mean Weighted Flow Time", Technical Report: UTDCS-1-87.

[44] Leung Joseph Y-T., Young G. H., "Minimizing Schedule Length Subject to Minimum Flow Time", Technical Report: UTDCS-9-87.

[45] Leung Joseph Y-T., Young G. H., "Minimizing Total Tardiness On a Single Machine with Precedence Constraints", Technical Report: UTDCS-4-89.

[46] Levine J., "An Efficient Heuristic Scheduler for Hard Real-Time Systems", Master's Thesis, Naval Postgraduate School, Monterey, California, Sept. 1991.

[47] Lie A. et al, "Change Oriented Versioning in a Software Engineering Database", Proceedings of 2nd International Workshop on Software Configuration Management, Princeton, New Jersey, Oct. 24,1989. pp. 56-65.

[48] Liu L., and Horowitz E., "Object Database Support for a Software Project Management Environment", Proceedings of the ACM SIGSOFT/SIGPLAN, Nov. 28-30, 1988. pp. 85-96.

[49] Lobba A., "Automated Configuration Management", Proceedings of IEEE conference on Software Tools 1987. pp. 100-103.

[50] Longstreet D. H., "Software Maintenance Management", Software Maintenance and Computers, IEEE Computer Society Press Tutorial, 1990. pp. 85-89.

[51] Luqi, "Software Evolution Through Rapid Prototyping", IEEE Computer 22, 5. May 1989, pp 13-25.

[52] Luqi, "A Graph Model for Software Evolution", IEEE Transaction on Software Engineering. Vol. 16. NO. 8. Aug. 1990. pp. 917-927.

[53] Luqi and Ketabchi M., "A Computer-Aided Prototyping System", IEEE Software, Mar. 1988. pp. 66-72.

[54] Luqi and Berzins V., "Rapidly Prototyping Real-Time Systems", IEEE Software, Sept. 1988. pp. 25-36.

[55] Madhavji N., "The Prism Model of Changes", Proceedings of 13th International Conference on Software Engineering, May 13-17, 1991-Austin, Texas, USA. pp. 166-177.

[56] Moquin B., "Software Configuration Management Tools: Change Management vs. Change Control", Proceedings of IEEE Phoenix Conference on Computers and Communications, 1985, pp. 97-100.

[57] Mostov I., "A Model of Software Maintenance for Large Scale Military Systems", Master's Thesis, Naval Postgraduate School, Monterey, California, June. 1990.

[58] Mostov I., Luqi, and Hefner K., "A Graph Model for Software Maintenance", Tech. Rep. NPS52-90-014, Computer Science Department, Naval Postgraduate School, Aug. 1989.

[59] Narayanaswamy K. and Scacchi W., "Maintaining Configuration of Evolving Software System", IEEE Trans. on Software Eng. SE-13,3. Mar. 1987, pp. 324-334.

[60] Nelson et. al "Clustering, Concurrency control, Crash recovery, Garbage collection, and Security in OODBMS", NPS-Tech. Report Feb. 1991. pp. 5-

[61] Nester J. R., "Toward a Persistent Object Base" Technical Memorandum, SEI-86-TM-8, Software Engineering Institute, Caregie-Mellon University July 1986

[62] Nester J. R., "Views for Evolution in Programming Environments", Presented at FeaCase '89, Integrated Data Management for Software Engineering, Nov 1989.

[63] "Ontos DB 2.2 Reference Manual, Volume 1 Class and Function Libraries", Ontos INC. Release 2.2, 1992.

[64] "Ontos DB 2.2 Developer' s Guide", Ontos INC. Release 2.2, 1992

[65] Perry D. E., and Kaiser G. E., "Infuse: A Tool for Automatically Managing and Coordinating Source Changes in Large Systems", Proceedings of the 1987 ACM Fifteenth Annual Computer Science Conference. St Louis, Missouri, February 1987. pp 292-299.

[66] Perry D. E., and Kaiser G. E., "Models of Software Development Environments", IEEE Transactions on Software Engineering, Vol. 17, NO. 3, March 1991.

[67] Ramamritham K., Stankovic J. A., Shiah P., "Efficient Scheduling Algorithm for Real-Time Multiprocessor Systems", COINS Technical Report 89-37, Dept. of Computer and Information Science, University of Massachusetts, 1989.

[68] Rullo T. A., "Advances in Computer Programming Management", Vol. 1. Heyden Advances Library in EDP Management, 1980. pp. 152-186.

[69] Schneidewind N. F., "The State of Software Maintenance", IEEE Transaction on Software Engineering. Vol. SE-13, NO. 3. March 1987. pp. 303-310.

[70] Simmonds Ian, "Configuration Management in the PACT Software Engineering Environment", Proceedings of 2nd International Workshop on Software Configuration Management, Princeton, New Jersey, Oct. 24,1989. pp. 118-121.

[71] Sommerville Ian "Software Engineering", Fourth edition, Addison-Wesley 1992

[72] Stankovic J. A., Ramamritham K., Shiah P., and Zhao W., "Real-Time Scheduling Algorithms for Multiprocessors", COINS Technical Report 89-47.

[73] Thomas Ian, Penedo M. H., and Ploedereder E., "Object Management Issues for Software Engineering Environments - Workshop Report-", Proceedings of the ACM SIGSOFT/SIGPLAN, Nov. 28-30, 1988. pp. 226-234.

[74] Thomas Ian, "Version and Configuration Management on Software Engineering Database", Proceedings of 2nd International Workshop on Software Configuration Management, Princeton, New Jersey, Oct. 24,1989. pp. 23-25.

[75] Tichy W. F., "RCS- A System for Version Control", Software Practice and Experience, VOL. 15 (7), July 1985. pp 637- 654.

[76] Tichy W. F., "Tools for Software Configuration Management", International Workshop on Software Version and Configuration Management", Grassau, FRG 27-29 January 1988.

[77] Van Tilburg R. L., "Software Configuration Management An Update", Proceedings of IEEE Phoenix Conference on Computers and Communications, 1985, pp. 97-100.

[78] William B. Franks, C. J. Fox, and B. A. Nejmeh, "Software Engineering in the UNIX/C Environment", Prentice Hall 1991

[79] Won Kim and Lochovsky F. "Object-Oriented Concepts, Databases, and Applications", ACM Press, Addison-Wesley, 1989.

[80] Ware Myers, "Allow Plenty of Time for Large-Scale Software", IEEE Software, July 1989.

[81] Silberschatz A., Stonebraker M., and Ullman J., "Database Systems: Achievements and Opportunities", Communication of the ACM, October 1991/Vol. 34, No. 10, pp. 110-120.

[82] Xu Jia, Parnas D., " Scheduling Processes with Release Times, Deadlines, Precedence, and Execlution Relations", IEEE Transactions on Software Engineering, Vol. 16, No. 3, March 1990.

[83] Xu Jia., "On Satisfying Timing Constraints in Hard-Real-Time Systems", IEEE Transactions on Software Engineering, Vol. 19, No. 1, January 1993.

[84] Xu Jia., "Multiprocessor Scheduling of Processes with Release Times, Deadlines, Precedence, and Exclusion Relations", IEEE Transactions on Software Engineering, Vol. 19, No. 2, February 1993.

[85] Zdonik Stanley B. "Version Management in an Object-Oriented Database" Proceedings of an international workshop, Trondheim, Norway, June 1986, pp.405-422 .

[86] Zhao W., Ramamritham K., Stankovic J. A., "Scheduling Tasks with Resource Requirements in Hard Real-Time Systems", IEEE Transactions on Software Engineering, Vol. SE-13, NO. 5, May 1987, pp. 564-576.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Dudley Knox Library Code 52 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chairman, Department of Computer Science Code CS Naval Postgraduate School Monterey, California 93943	1
4.	Computer Technology Programs Code 37 Naval Postgraduate School Monterey, California 93943	1
7.	Military Technical College (Egypt) c/o American Embassy (Cairo, Egypt) Office of Military Cooperation Box 29 (TNG) FPO, NY 09527-0051	2
8.	Military Research Center (Egypt) c/o American Embassy (Cairo, Egypt) Office of Military Cooperation Box 29 (TNG) FPO, NY 09527-0051	2
8.	Armament Authority-training Department (Egypt) c/o American Embassy (Cairo, Egypt) Office of Military Cooperation Box 29 (TNG) FPO, NY 09527-0051	5

9. Professor Vadis Berzins, Code CS/Bz — 2
Department of Computer Science
Naval Postgraduate School
Monterey, California 93943

10. Professor Luqi, Code CS/Lq — 2
Department of Computer Science
Naval Postgraduate School
Monterey, California 93943

11. Professor Jon Butler, Code EC — 1
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, California 93943

12. Professor Mantak Shing — 1
Computer Science Department, Code CS
Naval Postgraduate School
Monterey, CA 93943

12. Professor Yuh-Jeng Lee — 1
Computer Science Department, Code CS
Naval Postgraduate School
Monterey, CA 93943

13. Professor R. B. Mcghee — 1
Computer Science Department, Code CS
Naval Postgraduate School
Monterey, CA 93943

14. Professor Tarek Abdel-Hamid — 1
Administrative Science Department, Code AS
Naval Postgraduate School
Monterey, CA 93943

15. Colonel Salah M. Badr — 5
Egyptian Armament Authority - Research Department
c/o American Embassy (Cairo, Egypt)
Office of Military Cooperation
Box 29 (TNG)
FPO, NY 09527-0051

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

3 2768 00311301 0